UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Análisis de métricas ligeras indicadoras de la calidad en Internet

Máster Universitario en Investigación e Innovación en Tecnologías de la Información y las Comunicaciones (i2-TIC)

Autor: Miravalls Sierra, Eduardo
Tutor: Aracil Rico, Javier

Junio, 2017

ANÁLISIS DE MÉTRICAS LIGERAS INDICADORAS DE LA CALIDAD EN INTERNET

Autor: Miravalls Sierra, Eduardo
Tutor: Aracil Rico, Javier

HPCN Research Group
Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid

Junio, 2017

Agradecimientos

En primer lugar, me gustaría agradecer al Ministerio de Educación Cultura y Deporte por concederme la beca de colaboración para este curso.

A mis compañeros del laboratorio, en especial a David y Javier, que me han ayudado a centrar el tiro y a terminar este trabajo.

A mi tutor Javier Aracil y al grupo HPCN por dejarme trabajar para ellos un año más.

Y a mis padres y mi hermana, que me han seguido aguantado otro año más.

Time, Dr. Freeman? Is it really that time again?
It seems as if you only just arrived. You've done a great deal in a small time span.
You've done so well, in fact, that I've received some interesting offers for your services.
Ordinarily, I wouldn't contemplate them... but these are extraordinary times.

Rather than offer you the illusion of free choice, I will take the liberty of choosing for you... if and when your time comes round again.
I do apologize for what must seem to you an arbitrary imposition, Dr. Freeman.
I trust it will all make sense to you in the course of... well...
I'm really not at liberty to say. In the meantime... this is where I get off.

G-Man

Abstract

Abstract —

The number of devices connected to the Internet is growing exponentially with the now ubiquitous Internet of Things. Each new connected device demands network resources to carry out its tasks, and further stresses the infrastructures. In order to be able to cater for this devices, we need to be able to measure how they perceive the network, and if the Quality of Service Degrades past a breaking point, some applications might stop working. Thus, network monitoring is an important task for network operators. However, the usual model of a single high performance probe in an aggregation node does not seem to scale as the traffic rates are growing more quickly than the technology.

Hence, it is interesting to deploy metrics for the quality of service in small embedded devices or even in the routers themselves, which would monitor smaller networks with lower traffic rates. These metrics could be collected periodically in an aggregation node, to make big picture analysis and take proactive actions to prevent quality of service degradations. However, these embedded devices usually have very small memory and computational resources. Thus, it is important to develop lightweight metrics that require to store little state in memory and only perform some simple computations per packet.

In this thesis we explore the capabilities of a router and an ODROID C2 have as active measurement probes. Afterwards, some experiments are performed to determine the suitable amount of traffic an ODROID could capture in a local network. Then, we explore the resources required to detect TCP retransmissions in Internet flows, providing models for both the accuracy and the memory requirements of the proposed algorithms. Finally, these algorithms are tested versus Tshark with real enterprise traces and the predictions derived from the presented models are also validated.

Key words — Monitoring, active measurements, passive measurements, capture engines

Resumen

Resumen —

Con la llegada del Internet of Things (IoT) ha crecido de manera exponencial el número de dispositivos conectados a Internet, cada uno de ellos requiriendo distintas calidades de servicio mínimas para poder funcionar de manera satisfactoria para los usuarios. Esto pone de manifiesto la importancia de la monitorización de las redes, y de la aplicación proactiva de políticas que eviten que la calidad de servicio se degrade demasiado. Los procesos de monitorización y captura de alto rendimiento tienden a centralizarse en un solo nodo en el que se analice todo el tráfico, sin embargo, éstos sistemas no pueden escalar hasta tasas arbitrarias.

Por tanto, es interesante analizar la posibilidad de trasladar estos procesos de monitorización a los dispositivos IoT de una red o incluso en los propios routers, llevando este proceso a los extremos de la red, donde la tasa y el nivel de agregación son menores. Sin embargo, estos dispositivos tienden a estar muy limitados en recursos computacionales y de memoria disponible, por lo que sería beneficioso estudiar si las métricas utilizadas pueden simplificarse para consumir menos recursos, aunque sea a costa de perder un poco de precisión.

En este trabajo se estudian las capacidades como dispositivos de medida activa de un router y una ODROID C2, y se realizan experimentos para ver el impacto de utilizar canales inalámbricos. También se estudian las capacidades de la ODROID como motor de captura en una red local. A continuación se estudian mecanismos simplificados de detección de retransmisiones en flujos TCP adecuados para ser desplegados en dispositivos que no puedan mantener mucho estado. Se presentan modelos analíticos con los que predecir la tasa de error y de consumo de memoria de los algoritmos, y se validan con trazas con tráfico real tomado en entornos empresariales frente a Tshark.

Palabras clave — Medidas activas, medidas pasivas, monitorización, motores de captura

Glosario

Jitter Medida de la variabilidad de la latencia de una comunicación. 3, 10, 12

Key Performance Indicator paŕametro clave para medir las prestaciones de un servicio. 1

Lockless FIFO cola de mensajes que permite la concurrencia sin necesidad de utilizar semáforos para gestionarla, debido a que está implementada solo con operaciones atómicas. 9

Polling proceso de consulta síncrona del estado de un dispositivo por una aplicación o dispositivo. Este proceso es opuesto al mecanismo de interrupciones, por el cual es el propio dispositivo el que notifica (mediante una interrupción) cuando ocurre un cambio de estado o evento. 9

Redundant Array of Independent Disks es una tecnología para utilizar varios discos lógicos como uno solo virtual. Existen diferentes configuraciones para añadir redundancia a los datos (a costa de perder capacidad de almacenamiento) que permita recuperarlos en caso de que uno de los discos falle. En este trabajo se refiere a la configuración RAID 0, que brinda el máximo rendimiento y capacidad pero a costa de perder toda la redundancia. 2

Switch Port ANalyzer mecanismo de monitorización de tráfico que consiste en que se reenvíe por uno de los puertos todo el tráfico agregado que se recibe por las otros puertos. 1

Ancho de Banda máxima cantidad de información por unidad de tiempo que un canal es capaz de transportar. 11

Core aislado el kernel de Linux permite mediante el parámetro `isolcpus` indicar los *cores* que el planificador de procesos debe ignorar. En estos *cores* solo podrán ejecutarse aplicaciones a las que se les haya fijado la afinidad a ese *core* con la utilidad `taskset`[1] o con `pthread_setaffinity_np()`[2]. 9

Flujo TCP conjunto de paquetes TCP que comparten una misma 4-tupla y una cierta localidad temporal. 25

[1] `linux.die.net/man/1/taskset`
[2] `linux.die.net/man/3/pthread_setaffinity_np`

Flujo TCP patológico dado un cierto umbral, se entiende que es un flujo TCP que tiene un porcentaje de retransmisiones por encima del mismo. En este trabajo se ha fijado el umbral en 5 %, y se ha requerido además que la sesión TCP tenga al menos 100 paquetes. 25, 35

Latencia es el tiempo que trascurre desde que un byte sale de un extremo de la conexión hasta que llega al otro extremo. 11

Motor *One-Copy* Motor de captura que copia los paquetes de los anillos de la tarjeta de red a un *buffer* intermedio al que pueden acceder las aplicaciones cliente. 5, 8, 23

Motor *Zero-Copy* Motor de captura que permite leer desde el espacio de usuario la memoria de la tarjeta de red. 5, 7, 23

Motor de Captura *(traffic sniffer)* es un programa especialmente diseñado para leer los paquetes de una o varias interfaces de red, y que provee una API para que las aplicaciones cliente puedan procesar dicho tráfico. 5

Número de Secuencia entero de 32 bits que TCP utiliza para identificar cada byte de datos que debe transportar. 26

PCAP Packet CAPture, formato de almacenamiento de paquetes de red en ficheros, llamados trazas. 6

Pérdidas paquetes que no llegan al otro extremo, por el motivo que sea. Estos paquetes pueden descartarse porque no pasen alguna suma de control (checksum/CRC), porque un *buffer* esté lleno, etc. 12

Sesión TCP una sesión entre dos *hosts* consta de dos flujos TCP, uno por cada sentido. El cliente es el *host* que envió el primer SYN, y al otro *host* se le denomina Servidor. 26

Acrónimos

API *Application Program Interface*. 5–7

BPF *Berkeley Packet Filter*. 6

CAPEX *CAPital EXpenditure*. 2

COTS *Commercial Off-The-Shelf*. 2, 3, 13

DPDK *Data Plane Development Kit*. 9

EAL *Environment Abstraction Layer*. 9

FNR *False Negative Rate*. 37

FPR *False Positive Rate*. 37

HPC *High Performance Computing*. 1

HPET *High Precision Event Timer*. 9

HTTP *HyperText Transfer Protocol*. 34

IoT *Internet of Things*. 1

IP *Internet Protocol*. 11

IPPM *IP Performance Metrics*. 10

ISP *Internet Service Provider*. 1

ITO *Inactivity Time-Out*. 33

KPI *Key Performance Indicator*. 1, 3, 10, 25, *Glossary: Key Performance Indicator*

MSL *Maximum Segment Lifetime*. 33

NFV *Network Function Virtualization*. 2, 27

NIC *Network Interface Card*. 5, 7, 9

OPEX *OPerational EXpenditure*. 2

OWD *One Way Delay*. 11

PPV *Positive Predictive Value*. 36, 37

QoS Calidad de Servicio (*Quality of Service*). 1, 3, 15

RAID *Redundant Array of Independent Disks*. 2, *Glossary: Redundant Array of Independent Disks*

RSS *Receive Side Scaling*. 7

RTT *Round Trip Time*. 11, 12

SDN *Software Defined Networking*. 2

SNMP *Simple Network Management Protocol*. 2

SPAN *Switch Port ANalyzer*. 1, 26, *Glossary: Switch Port ANalyzer*

TCP *Transmission Control Protocol*. 10, 25

TNR *True Negative Rate*. 36, 37

TPR *True Positive Rate*. 36, 37

TWAMP *Two Way Active Measurement Protocol*. 10

UDP *User Datagram Protocol*. 11

Índice general

Índice de tablas

Índice de figuras

1

Introducción

Analizar las prestaciones de una red de ordenadores es un problema complejo, incluso hoy en día. El número de dispositivos conectados a Internet sigue un crecimiento exponencial, en especial con el auge del *Internet of Things* (IoT). Además, hay multitud de aplicaciones que requieren enviar sus datos por la red y, dependiendo del tráfico que se esté enviando y de lo crítico que sea, puede que a los usuarios les interese contratar con su *Internet Service Provider* (ISP) una garantía de que ciertas características de la red no bajarán de ciertos umbrales, para garantizar cierta Calidad de Servicio (*Quality of Service*) (QoS).

Es de especial interés tanto para el cliente como para el ISP, poder medir el QoS y detectar cuándo éste se degrada para poder poner en funcionamiento medidas proactivas que eviten que se degrade demasiado. Es más, cuando hay un problema en una red, con la gran variedad de dispositivos que se pueden encontrar, y puede que estén incluso separados geográficamente en caso de grandes redes como las de un ISP, los administradores de red no pueden ir comprobando dispositivo por dispositivo. Por tanto habitualmente se utilizan mecanismos de análisis de la salud de la red que puedan analizar varios equipos, o redes enteras, en un solo punto.

Habitualmente se despliegan uno o varios puntos de medida en la red, mediante una sonda de red, que se encarga de capturar el tráfico de red, recoger estadísticas y computar métricas indicadoras *Key Performance Indicator* (KPI) de la salud de la red.

Para niveles de agregación muy elevada o redes de muy alto rendimiento, con millones de flujos concurrentes y con anchos de banda de 10, 25, 40 ó 100 Gb/s, es necesario desplegar sistemas de alto rendimiento (*High Performance Computing* (HPC)) como el M^{3}Omon [1], conectados a un nodo de agregación de la red mediante un *Switch Port ANalyzer* (SPAN) como se ilustra en la Figura 1.1.

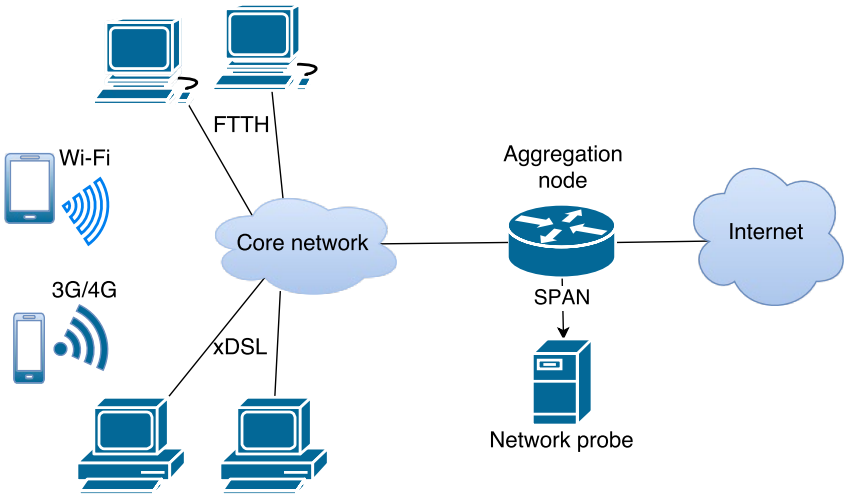


Figura 1.1: Un escenario típico de monitorización.

Ser capaz de analizar tales volúmenes de datos (Big Data) mediante técnicas de Machine Learning o de análisis estadísticos más sencillos, es todavía un problema de ingeniería abierto, en el que se está investigando activamente [2, 3] y hay numerosos estándares donde se dan recomendaciones acerca de cómo hacerlo [4–7].

No hay que olvidar el factor económico y, a pesar de que se utilice equipamiento *Commercial Off-The-Shelf* (COTS), los operadores quieren reducir su *CAPital EXpenditure* (CAPEX) y *OPerational EXpenditure* (OPEX) lo máximo posible, y las soluciones que requieren comprar hardware adicional como GPUs o FPGAs tienden a ser muy costosas [8–11]. Además, poder almacenar todo ese tráfico para realizar análisis offline es un problema complicado [12], debido a que los anchos de banda que proporcionan varios discos en una configuración *Redundant Array of Independent Disks* (RAID) no es suficiente cuando el ancho de banda pasa de los 10 ó 25Gb/s, ya que el planteamiento actual no parece escalar linealmente [13].

Sin embargo, se le puede dar un enfoque distinto a la monitorización. Las redes constan de multitud de routers, los cuales muchos llevan ya un Linux embebido que puede ser programable por el administrador de red. Sería deseable poder realizar estos análisis de red de manera más local donde los propios dispositivos embebidos del IoT o los propios routers sean capaces de analizar la porción del tráfico que ven. Éstos dispositivos podrían calcular esas mismas métricas, u otras métricas más simples que provean la misma información, y utilizando por ejemplo protocolos como *Simple Network Management Protocol* (SNMP) o mecanismos de generación de flujos como OpenFlow o IPFIX [14], éstas estadísticas se podrían recoger en un nodo de agregación para ser analizadas en su conjunto. Esto permite reducir el coste del despliegue del sistema de monitorización de red, ya que reaprovecha el propio hardware y solo requiere soluciones software. Alejarnos del hardware concreto desplegado, manejar las redes y controlarlas por medio de software es lo que se llama *Software Defined Networking* (SDN) [15] que ejecuta funciones virtuales de red *Network Function Virtualization* (NFV) [16] que encapsulan una funcionalidad o aplicación, como sería la monitorización de red, para reducir los costes [17]. Por tanto, es interesante analizar mecanismos ligeros de monitorización que se puedan desplegar en dispositivos embebidos.

1.1. Alcance

- Para las medidas activas, se ha limitado a los principales KPIs detallados en el Capítulo 2, ya que son las métricas más importantes para evaluar el QoS.
- Como punto de partida para el análisis de métricas pasivas ligeras en el Capítulo 5, se ha partido de un Trabajo Fin de Grado previo [18]. Los algoritmos se han formalizado y generalizado, se han propuesto modelos sobre su funcionamiento y se han evaluado con trazas con tráfico real.
- Se ha limitado el estudio a tráfico IPv4, ya que el tráfico IPv6 todavía no es predominante en las redes comerciales.
- Para el Capítulo 5, en ocasiones puede ocurrir que haya paquetes duplicados en la red que si no se tienen en cuenta, pueden alterar las medidas pasivas. Se asumirá que no hay paquetes duplicados debido a que es un problema ya estudiado [19] y existen mecanismos para eliminarlos como paso previo antes de analizar el tráfico.

1.2. Objetivos

El objetivo general de este TFM es analizar nuevos enfoques basados en métricas ligeras para la evaluación de las prestaciones y QoS de redes de comunicaciones con el fin de mejorar la escalabilidad y disminuir el coste de las aproximaciones actuales por el uso de alternativas distribuidas y sistemas embebidos COTS. Para ello, se plantean los siguientes objetivos:

- Estudiar el estado del arte en torno al ecosistema de motores de captura y proceso, analizando tanto sus características como su adecuación para sistemas de cómputo general embebidos tipo ODROID, etc. que comienzan a proliferar en las redes de hoy en día, sin perjuicio de que se puedan considerar otros sistemas que los sustituyan o complementen.
- Analizar los principales KPIs (pasivos, activos o de cualquier otro género) de servicio de la red, tales como latencia, *jitter*, ancho de banda, pérdidas, u otros.
- Desarrollar KPIs simplificados y métodos para la evaluación del impacto de dichas simplificaciones en términos de ahorro de recursos y coste, y analizar las posibilidades de implantación de los mismos en sistemas embebidos COTS.

1.3. Estructura del documento

En el Capítulo 2 se presenta una revisión del estado del arte de motores de captura y de métricas indicadoras de la calidad.

En el Capítulo 3 se presentan unos experimentos realizados con un router y una ODROID como dispositivos de medida activa para determinar qué valores de estos KPIs perciben.

En el Capítulo 4 se presentan y analizan unos experimentos para explorar las capacidades de una ODROID como dispositivo de captura.

En el Capítulo 5 se presentan los algoritmos de detección de retransmisiones en flujos TCP, se analizan, modelan y evalúan.

Finalmente, en el Capítulo 6 se incluyen algunas reflexiones finales y líneas de trabajo futuro.

2

Estado del Arte

En este capítulo se revisa el estado del arte de los principales elementos de la monitorización, estos son: los principales motores de captura que existen para poder analizar el tráfico de manera pasiva; y las definiciones de las métricas indicadoras de la calidad que son de interés para este trabajo.

2.1. Motores de captura

Cuando se habla de diseño de un motor de captura se suelen distinguir ciertas características de su arquitectura.

Una característica importante es el número de copias intermedias, y por ello los motores se suelen clasificar en motores Cero-copia (Zero-copy) y en Uno-copia (One-copy). Por un lado, los motores de captura cero-copia permiten acceder directamente a la memoria interna de la tarjeta de red, *Network Interface Card* (NIC), desde el espacio de usuario para acceder a los paquetes. El objetivo es reducir el sobrecoste de hacer copias intermedias y tratar de reducir el número de llamadas al sistema. Tiene el inconveniente de que la tarjeta puede quedarse sin memoria si las aplicaciones cliente del motor de captura no liberan los recursos a la velocidad suficiente, por lo que se perderán paquetes. Por otro lado, los motores de captura Uno-copia copian los paquetes a un *buffer* intermedio, no necesariamente con el mismo formato que en las estructuras internas de la tarjeta de red, al cual pueden acceder sus clientes a través de su *Application Program Interface* (API). Esto tiene la ventaja de que este *buffer* puede ser mucho más grande que la memoria de la NIC, lo que permite amortiguar picos de carga en los que la tasa de paquetes por segundo sea mayor de lo que la aplicación es capaz de procesar, a costa de tener que hacer una copia intermedia.

2.1.1. TCPDump

TCPDump[1] [20, 21] es el estándar *de facto* cuando se habla de motores de captura. `tcpdump` permite capturar el tráfico y mostrar en tiempo real los paquetes diseccionados en formato texto por la salida estándar. Lo interesante es que también permite capturar directamente a un fichero pcap para poder analizar el tráfico de manera offline. TCPdump provee la librería libpcap como API C/C++ para poder capturar tráfico o analizar ficheros de traza en formato pcap. TCPDump permite aplicar al tráfico capturado filtros *Berkeley Packet Filter* (BPF), para prefiltrarlo y solo almacenar el subconjunto que nos interesa. La libpcap ha probado ser muy versátil, y se encuentra disponible tanto para sistemas Windows como Linux, y para plataformas Intel y ARM. Sin embargo, ha demostrado estar limitada [22] para la captura y almacenamiento de tráfico en entornos de alto rendimiento cuando la tasa supera el Gb/s.

Tradicionalmente TCPDump no ha sido capaz de capturar tráfico a alta tasa, y esto motivó la aparición de otros motores de captura, que traten de solventar los problemas de diseño que le impiden obtener mayor tasa en una configuración más general. Estas alternativas permiten evitar situaciones de pérdida de datos en captura, que pueden poner en peligro las medidas pasivas de calidad realizadas.

En [12] se muestra que al optimizar el rendimiento de `tcpdump` como motor de captura hay que estudiar detenidamente la arquitectura hardware de la máquina en la que se quiere capturar. Durante la configuración, hay que tener especial cuidado con dos factores: la planificación de los procesos y el número de colas de recepción. El primero consiste en fijar la afinidad del proceso de TCPDump a un *core* concreto, para evitar que el planificador pueda cambiarlo a otro *core* aleatorio durante la ejecución. Además, es vital fijar la afinidad a un *core* que se encuentre en el mismo nodo NUMA que la NIC, para obtener el máximo rendimiento. El segundo es fijar el número de colas de recepción que TCPDump utilice, asegurando además que su vector de interrupción esté conectado al mismo nodo NUMA. Por defecto TCPDump utiliza tantas colas como *cores* tenga el ordenador en el que se está ejecutando. Sin embargo, con 4 y con 5 colas de recepción utilizando el *driver* de Intel *ixgbe* y la libpcap se puede conseguir capturar todo el tráfico con tamaño de paquete mínimo a 10Gb/s.

`tcpdump` utiliza una herramienta de bajo nivel `dumpcap` [21] cuyo único objetivo es capturar los paquetes desde una interfaz de red para escribirlos en algún dispositivo de almacenamiento permanente.

2.1.2. Netsniff-ng

Netsniff-ng[2] [20, 21] es un conjunto de ocho utilidades diseñadas con el objetivo de cubrir todas las necesidades que un desarrollador o administrador de sistemas Linux puedan tener. Entre las utilidades encuentran el motor de captura (`netsniff-ng`), una generadora de tráfico multihilo (`trafgen`) y un compilador de filtros BPF, entre otras.

[1] `www.tcpdump.org`

`netsniff-ng` es un motor de captura del tipo zero-copy cuyo diseño trata de ser el de un motor de altas prestaciones. Al igual que con TCPDump, para lograr capturar paquetes a alta tasa se recomienda estudiar detenidamente la arquitectura concreta de la máquina en la que se quiere ejecutar, para configurar todos los elementos que intervienen en la captura (los parámetros del *kernel*, la configuración de red, el planificador de red, entre otros) de manera óptima.

2.1.3. Netmap/PFRING DNA

Netmap [23, 24] es un motor de captura Zero-copy cuyo diseño trata de aprovechar las arquitecturas multicore utilizando caminos de datos independientes entre procesos distintos. Para repartir el tráfico distribuye los paquetes entre diferentes colas utilizando la tecnología *Receive Side Scaling* (RSS). Netmap opera en dos modos, en uno de ellos se usa la pila de protocolos del *kernel* de manera normal, y el modo *netmap*, por el cual los paquetes no pasan por la pila de protocolos del *kernel*, y se dejan a disposición de las aplicaciones cliente directamente. Netmap implementa una serie de estructuras para hacer el acceso a las colas de la NIC trasparente, independientemente del tipo de tarjeta que se esté usando. Esta fina capa de abstracción le permite además soportar un gran número de NICs de diferentes fabricantes. El inconveniente que tiene es que la sincronización de las colas de netmap con las de la tarjeta es asíncrona, y es la propia aplicación la que debe desencadenarlo con una llamada al sistema. La ventaja es que mejora el rendimiento de la aplicación puesto que reduce el número de llamadas al sistema por paquete: por cada una de esas llamadas todo un lote de paquetes (*batch*) queda disponible a la vez.

Los mismos autores de Netmap han desarrollado PFRING DNA (*Direct NIC Access*) [23] como evolución de otro motor de captura (PFRING), el cual tiene soporte para plugins, y filtrado avanzado de paquetes además de BPF. PFRING DNA también es un motor zero-copy, pero a diferencia de netmap que no permite a sus aplicaciones clientes tener acceso directo a los registros de la tarjeta de red, PFRING DNA permite modificarlos directamente, saltándose completamente el *kernel*. Esto tiene la ventaja de aumentar el rendimiento al reducir el número de llamadas al sistema, pero tiene el inconveniente de que una aplicación podría introducir parámetros incorrectos a la tarjeta, lo que podría causar todo tipo de problemas. PFRING DNA separa la gestión de las colas del *kernel* y las traslada al espacio de usuario en cada aplicación cliente, de manera que las aplicaciones cliente son completamente independientes entre sí y el rendimiento puede escalar linealmente con cada *core*.

Tanto netmap como PFRING DNA implementan una API como la de la libpcap para poder acelerar aplicaciones que la usen sin necesidad de tener que modificarlas para que utilicen otras APIs.

[2] `http://netsniff-ng.org/`

2.1.4. PFQ

Paquet Family Queue (PFQ) [25] es otra infraestructura de captura de paquetes de alto rendimiento, que explota las ventajas de las arquitecturas multicore para poder capturar tráfico a alta tasa. Los autores abogan por utilizar un módulo del *kernel*, lo que los hace independientes de la tarjeta de red que se esté usando y solo requiere una pequeña modificación en los *drivers* para cambiar las llamadas al sistema por llamadas a funciones de PFQ, proceso que está completamente automatizado mediante *scripts*. El objetivo es saltar la pila del *kernel*, y permitir a PFQ la gestión completa del tráfico. PFQ es capaz incluso de acelerar *drivers* binarios, aunque en la práctica la mejora de rendimiento obtenida es mucho menor.

PFQ trata cada cola de una tarjeta de red como una fuente de datos independiente, y construye bloques funcionales, que son conjuntos de paquetes que provienen de una o varias fuentes de datos. Para construir estos bloques funcionales, si un paquete debe estar en varios bloques funcionales diferentes, se copia en paralelo a los dos, aunque con las cachés el impacto de múltiples copias es pequeño. Estos bloques funcionales se procesan utilizando un lenguaje funcional `pfq-lang`[3] por el motor funcional. Este motor funcional puede aplicar operaciones sobre los paquetes, que les añadan metadatos o tengan efectos como por ejemplo generar eventos de entrada-salida. Cada motor funcional ejecuta las acciones de sus paquetes en serie, pero todos los motores funcionales se ejecutan en paralelo. Finalmente, la salida del motor funcional se envían a uno o más *endpoint*, que puede ser una cola de un *socket* en una aplicación cliente o la propia pila de red del *kernel*.

PFQ provee de una API tanto de C/C++ como de Haskell para desarrollar aplicaciones que lo usen.

2.1.5. HPCAP

HPCAP es un motor de captura de alto rendimiento del grupo de investigación High Performance Computing and Networking (HPCN) desarrollado por Víctor Moreno durante su tesis doctoral [26]. Éste motor de captura One-copy está basado en el driver de Intel `ixgbe` y soporta tarjetas Intel de 10Gb/s. HPCAP provee de una API C similar a la `libpcap` para que las aplicaciones cliente puedan recibir los paquetes de red. Este motor tiene una herramienta cliente asociada, `hpcap-dd`, que es la encargada de escribir los paquetes a un dispositivo de almacenamiento permanente. Recientemente han publicado una nueva versión [27] desarrollada por Guillermo Julián Moreno que soporta las tarjetas de Intel y Mellanox de 40Gb/s.

HPCAP utiliza un solo anillo de descriptores en el que la tarjeta va escribiendo los paquetes. Para aprovechar las arquitecturas multicore, utiliza una arquitectura de múltiples escritores-múltiples consumidores donde varios hilos del *kernel* escriben los paquetes en un gran *buffer* intermedio del que consumen las aplicaciones clientes. Cuando tiene varios lectores, el espacio del *buffer* se consume al ritmo del más lento, de manera que el *buffer* podría llenarse (ocasionando pérdidas de paquetes) si uno de los lectores es especialmente lento. Por ello, el tamaño mínimo recomendado del *buffer* es de 1GB,

para poder soportar tanto ráfagas de paquetes como lectores que tengan un tiempo de proceso irregular por paquete. Para evitar comunicaciones entre los escritores, el anillo de descriptores se segmente de forma que cada hilo se encarga exclusivamente de su región del anillo, que es disjunta de todas las demás.

Una de las características únicas de HPCAP es que a todos los paquetes se les añade una marca de tiempo según llegan por la red. Este marcado de tiempos lo realiza la propia NIC en el caso de las tarjetas Mellanox, que es más preciso que el marcado *software* que hay que hacer en las tarjetas Intel.

2.1.6. DPDK

Data Plane Development Kit (DPDK) [28] es un *framework* diseñado por Intel para procesar tráfico con muy alta tasa desde espacio de usuario. Sin embargo, toda la arquitectura de procesado del tráfico recae sobre el programador. DPDK proporciona una abstracción del entorno, *Environment Abstraction Layer* (EAL), que es un conjunto de librerías específicas para la arquitectura de la máquina en la que se va a ejecutar la aplicación. Esta capa de abstracción incluye una serie de módulos con los que gestionar la reserva de memoria, las estructuras con los datos y metadatos de los paquetes, colas FIFO *lockless*, utilidades de depuración, gestión del *High Precision Event Timer* (HPET) o de acceso a recursos PCIE, etc.

Al igual que otras soluciones, DPDK trata de explotar las bondades de las arquitecturas multicore, buscando la escalabilidad y la extensibilidad con el número de cores. Entre otras técnicas, utiliza la estrategia de *polling* para evitar el sobrecoste de las interrupciones y se recomienda ejecutar cada hilo de DPDK en un core aislado. DPDK tiene bastantes ejemplos con los que se plantean diferentes arquitecturas, utilizando las diferentes librerías que proveen.

Una de ellas se basa en el reparto de los paquetes[4] [5], donde un hilo distribuidor lee los paquetes desde el anillo de recepción y los trata de repartir entre sus trabajadores. El distribuidor asegura que todos los paquetes con el mismo identificador RSS vayan al mismo trabajador en orden mientras éste no solicite más trabajo, para poder implementar aplicaciones que trabajen con flujos unidireccionales además de trabajar con paquetes individuales. Esto asegura que no se pueda procesar dos paquetes con el mismo identificador RSS en paralelo o fuera del orden de llegada. Sin embargo, una vez un trabajador pide al distribuidor más trabajo, el distribuidor asume que éste ha terminado con ese flujo, y que puede repartir paquetes con ese mismo identificador RSS a otro trabajador. La librería no impone ninguna ordenación entre paquetes con diferente identificador RSS o si los trabajadores se guardan copias internas para ser enviadas más tarde. Esto implica que si un trabajador t_1 no envía sus paquetes antes de solicitar más trabajo, podría ocurrir que se asigne ese identificador RSS a otro trabajador t_2 que envíe sus paquetes antes que t_1, introduciendo desorden entre esos paquetes.

[3] `https://www.slideshare.net/nicolabonelli/functional-approach-to-packet-processing`
[4] `http://dpdk.org/doc/guides/prog_guide/packet_distrib_lib.html`
[5] `http://dpdk.org/doc/api/rte__distributor_8h.html`

2.2. Medidas indicadoras de la calidad

En esta sección incluimos algunos preliminares referentes a las medidas de prestaciones de red. En primer lugar, revisamos diversas propuestas de metodologías y buenas prácticas, así como sus limitaciones tanto en términos de medidas en general como en el contexto de conexiones inalámbricas. Por otro lado, proporcionamos las definiciones que vamos a manejar de los KPIs utilizados habitualmente por las operadoras, extraídas de este primer bloque de propuestas. Específicamente, por su relación con la QoE, vamos a considerar el ancho de banda, la latencia, el *jitter* y las pérdidas.

2.2.1. Metodologías de medida

Un problema recurrente a la hora de medir las prestaciones de una red, es que las medidas son muy sensibles de la metodología utilizada. Es por ello que surgieron iniciativas como *IP Performance Metrics* (IPPM) [29] o el *Two Way Active Measurement Protocol* (TWAMP) [30], que intentan fijar buenas prácticas y definiciones para las métricas, de manera que sean claramente objetivas y repetibles. Así, se trata de evitar resultados no repetibles debido a, entre otras cosas, componentes estocásticas no controladas en los modelos, errores de medida debidos a la metodología y diseño experimental, etc.

Pese a estos intentos, varios estudios muestran que se pueden obtener diferentes resultados para la misma métrica, no siempre equivalentes, dependiendo de la herramienta utilizada. Por ejemplo, el reciente estudio incluido en [31] muestra las divergencias entre algunas herramientas populares, ampliamente utilizadas por usuarios finales. Además, diversos factores relacionados con el propio sistema desde el que se realizan las medidas (p.e. carga de CPU, política del planificador, etc.) tienen efectos significativos sobre los resultados de las medidas [2].

Cuando se trata de medir las capacidades de un canal inalámbrico la tarea se complica, ya que este tipo de enlaces tiende a introducir pérdidas o latencias aleatorias que son difíciles de predecir, y que dependen de muchos factores. Con frecuencia se publican nuevos estudios en los que se intenta modelar el retardo de enlaces inalámbricos como en [32,33]. En ambos estudios recurren a modelos de tipo árbol de decisión (*decision tree*) o *random forest*, que son interpretables.

Otra complicación que pueden introducir los accesos inalámbricos es una pérdida notable de rendimiento cuando hay muchos usuarios [34] debido a las interacciones entre las peculiaridades del nivel de enlace y los protocolos de nivel superior.

Por ejemplo, *Transmission Control Protocol* (TCP) puede aumentar las retransmisiones como consecuencia de la variabilidad de la latencia por las colisiones a nivel de enlace, sin que dichas retransmisiones fuesen realmente necesarias. Como consecuencia, la evaluación de accesos que vayan a ser compartidos por un gran número de usuarios debe considerar estos casos a la hora de evaluar las prestaciones esperadas, lo que justifica la conveniencia de abaratar los dispositivos de medida utilizados.

2.2.2. Ancho de banda

El ancho de banda de un enlace mide la cantidad de información por unidad de tiempo que es capaz de transportar (p.e. en bits por segundo). En la práctica el término "ancho de banda" puede referirse a uno de varios conceptos [35, 36].

Dado un camino extremo a extremo formado por un conjunto ordenado de n enlaces $i = 1, ..., n$, se define la capacidad del enlaces i como la máxima tasa de transmisión a nivel *Internet Protocol* (IP), B_i. Por tanto, la capacidad B^* del camino se define como el mínimo de las capacidades $\{B_i\}_1^n$ de cada uno de los enlaces que forman el camino

$$B^* = \min_{i=1...n} \{B_i\} \tag{2.1}$$

En un camino los enlaces i_k tales que $B_{i_k} = B^*$ son denominados *narrow link*s o enlace angosto.

El ancho de banda disponible, *available bandwidth*, de un camino se define como la capacidad no usada en ese instante de tiempo. Es la métrica complementaria del ancho de banda consumido. Depende de B^* y de la cantidad de tráfico que esté circulando en ese momento por la red.

Por otro lado, el *Bulk Transfer Capacity* (BTC) es la máxima cantidad de información que un protocolo que implemente control de la congestión, p.e. TCP, puede enviar por unidad de tiempo.

Cuando se realizan medidas activas, según el protocolo de transporte utilizado se medirán cosas diferentes. La técnica de trenes de paquetes utiliza *User Datagram Protocol* (UDP) como protocolo de transporte, que permite medir la capacidad B^* del camino [2]. En cambio si se utiliza el protocolo TCP se mide el BTC de un camino.

Finalmente, se puede medir el ancho de banda consumido de manera pasiva contabilizando cuántos bytes por unidad de tiempo están transmitiendo por un enlace una aplicación o conjunto de equipos específicos.

2.2.3. Latencia

La latencia es el tiempo que transcurre desde que un byte es enviado hasta que llega al otro extremo.

Se distinguen la latencia en un solo sentido [6], *One Way Delay* (OWD), y la suma de las latencias en los dos sentidos, el tiempo de ida y vuelta, *Round Trip Time* (RTT). Estimar el OWD de una comunicación es difícil, ya que requiere que los relojes de los dos extremos se mantengan sincronizados. Por esta razón, se suele medir el RTT, que sortea el problema de sincronización de relojes ya que la base de tiempos es la misma. Asumiendo que las dos latencias en un solo sentido sean iguales, el OWD se estima en ocasiones como la mitad del tiempo que transcurre entre una petición y su respuesta, como se sugiere en protocolos como Q4S [3].

La forma más sencilla de estimar el RTT es realizando medidas activas con un mecanismo similar a un `ping` [37]. Estimar la latencia de manera pasiva monitorizando conexiones TCP requiere conocer los protocolos de nivel superior que se están utilizando para poder identificar peticiones con respuestas. Además, algoritmos como el de Nagle o el asentimiento retrasado (*delayed ACK*) dificultan la estimación del RTT debido a que las respuestas pueden no ser inmediatas, sobreestimando la latencia. Una forma inequívoca de identificar una petición y su respuesta en TCP es durante el establecimiento de la conexión, donde se puede estimar el RTT como la diferencia de tiempos entre el SYN y el ACK en el lado del servidor:

$$RTT = t_{ACK} - t_{SYN} \tag{2.2}$$

y de SYN,ACK y el ACK en el cliente:

$$RTT = t_{SYN,ACK} - t_{SYN} \tag{2.3}$$

2.2.4. Variación de la latencia

La variación de la latencia o *jitter* es un parámetro importante de la calidad, ya que afecta gravemente a aplicaciones multimedia tales como Voz sobre IP (VoIP).

Existen multitud de definiciones de *jitter*. La definición que vamos a utilizar es la dada en [4] y de manera equivalente en [3]: dadas $N+1$ medidas de latencia unidireccionales, $\{l_i\}_{i=0}^{N}$ podemos calcular N diferencias, $\{\Delta_j\}_{j=1}^{N}$

$$\Delta_j = |l_j - l_{j-1}| \tag{2.4}$$

y definir el *jitter* mediante un estadístico de centralidad de los $\{\Delta_j\}$ como la media aritmética o la mediana. Otras alternativas son estimar el *jitter* como el resultado de aplicar un filtro exponencial [4] de parámetro 1/16 a los $\{\Delta_j\}$; o estimarlo como la desviación estándar de las $\{l_i\}$.

2.2.5. Pérdidas

Las pérdidas son un indicador de saturación de la red entre otros, ya que los *routers* y los equipos finales descartan paquetes cuando reciben paquetes más rápido de lo que pueden procesar. También pueden perderse paquetes cuando los datos se corrompen, ya sea por un problema de *hardware* o por ruido en la comunicación, muy habitual en enlaces inalámbricos.

Los protocolos de medida Q4S o IPPM [7] utilizan números de secuencia para estimar las pérdidas. Esta estimación se calcula como el ratio de paquetes que no se recibieron frente al total esperado.

3

Medidas activas con dispositivos COTS de bajo coste

En este capítulo se pretende estudiar la viabilidad como dispositivos de medida activa dispositivos COTS de muy bajo coste como una ODROID o un router al que se le pueda instalar una distribución Linux. Para ello, se ha realizado una serie de experimentos con los que emular distintos escenarios de redes en entornos doméstico o de oficina y se han extraído algunas conclusiones.

3.1. Descripción del entorno experimental

Para la realización de los experimentos se ha utilizado el siguiente equipamiento:

- Dos routers TP-Link Archer C7 AC 1750 con una configuración MIMO 3×3×3. Uno de ellos se ha dejado con el *firmware* de fábrica[1], que se denotará como "TP-Link"; y en el segundo se ha instalando una distribución Linux dd-wrt[2], al que se denotará como "dd-wrt".

- PC1: un equipo de sobremesa con un procesador Intel Core i7 CPU 860 a 2.80GHz, 8GB DDR3 a 1333MHz, y dos interfaces de red: la integrada en la placa de intel Gigabit Ethernet (conectada a Internet) y una segunda interfaz utilizada para conectarse al router que proporciona una tarjeta de red PCI Broadcom Corporation NetXtreme II BCM5709 Gigabit Ethernet (rev 20). El sistema operativo era Ubuntu 14.04 Mate.

[1] 3.15.1 Build 160616 Rel.44182n

[2] `http://dd-wrt.com/site/index`

Tabla 3.1: Herramientas utilizadas.

Herramienta	Protocolo	Ancho de banda	Latencia	*Jitter*	Pérdidas
iperf3	TCP	✓	✗	✗	✓
iperf3	UDP	✓	✗	✓	✓
ping	ICMP	✗	✓	✗	∼

- PC2: otro equipo de sobremesa que tiene un Intel Core i7-2600 CPU a 3.40GHz, 8 GB de RAM DDR3 a 1333MHz, y se ha utilizado la interfaz de red integrada Intel Gigabit Ethernet. El sistema operativo era CentOS 7.
- Y por último, se ha utilizado una ODROID C2[3] con un adaptador de red WiFi 802.11n USB Edimax EW-7811UN que alcanza hasta los 150Mb/s.

Para realizar las medidas se han utilizado varias herramientas del estado del arte: `iperf3` para medir el ancho de banda, las pérdidas y el *jitter*; y `ping` para medir la latencia, ya que `iperf3` no proporciona esa métrica.

`iperf3` es la herramienta estándar *de facto* para medir la capacidad de un canal. `iperf3` tiene diferentes comportamientos en función del protocolo de transporte que se quiera utilizar. Utilizando TCP, es capaz de estimar el ancho de banda útil máximo; sin embargo, cuando se utiliza UDP es necesario indicar la tasa a la que se desea enviar, ya que al no tener retroalimentación del otro extremo, `iperf3` envía siempre a máxima tasa, y solo el servidor es capaz de detectar cuántos paquetes llegan realmente. Por ello, para medir el ancho de banda con UDP en cada escenario, se ha buscado primero de manera iterativa el ancho de banda máximo que se podía lograr sin pérdidas antes de tomar las medidas. Por tanto, es importante observar los valores calculados en el servidor siempre, ya que interesa estudiar los efectos que haya podido tener la red sobre el tráfico.

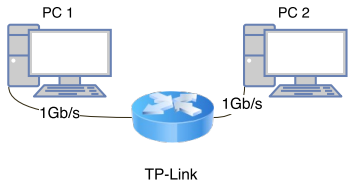
Para evitar los efectos del planificador, se ha fijado la afinidad de `iperf3` a un mismo *core* tanto en el cliente como en el servidor, aislado en el arranque para evitar que la competición por los recursos de CPU pudieran interferir en las medidas.

Se tomaron 10 medidas con `iperf3` por cada sentido en cada escenario, lo que nos proporciona un total de 100 muestras por sentido gracias a que cada ejecución de `iperf3` proporciona 10 muestras de 1 segundo. Del mismo modo, hemos realizado 100 medidas con `ping` por sentido para estimar la latencia de cada enlace. El uso de CPU fue siempre inferior al 20 %, por lo que no fue un factor limitante en nuestras pruebas. En todos los casos las pérdidas fueron nulas o menores del 1 % de los paquetes.

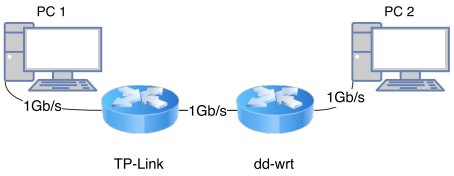
En la Tabla 3.1 se muestran las distintas configuraciones utilizadas y los datos que proporcionan cada una de ellas. Cabe destacar que aunque se puede utilizar `ping` como estimador de pérdidas, debido a que solo envía un paquete por segundo, es más bien una medida instantánea de conectividad [38] [4]. Salvo en canales con muchas pérdidas,

[3] `http://www.hardkernel.com/main/products/prdt_info.php`

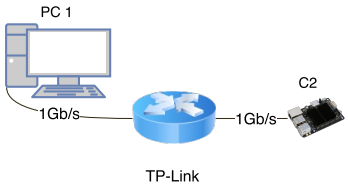
[4] Siempre que el otro equipo no tenga configurado que ignore los mensajes ICMP *echo requests*



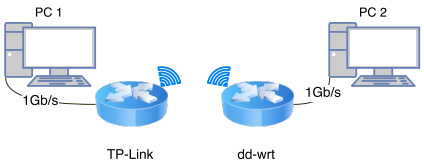
(a) Escenario 1



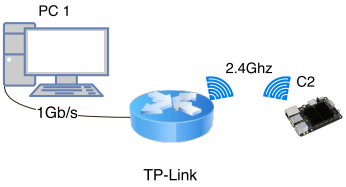
(b) Escenario 2



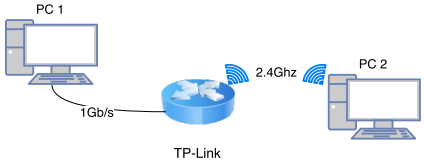
(c) Escenario 3



(d) Escenarios 4 y 5 con red a 2.4GHz, 8 y 9 a 5GHz



(e) Escenario 6



(f) Escenario 7

Figura 3.1: Topologías de red para cada escenario de medida activa.

con pérdidas esporádicas[5] que justo afecten al paquete ICMP, o que la red esté muy congestionada, normalmente el *echo REPLY* llegará de vuelta.

Se han diseñado varios escenarios experimentales en los que los únicos equipos conectados a esa red son un equipo que hace las veces de cliente y otro que hace las veces de servidor. Las medidas se han realizado en un laboratorio universitario, en escenarios cableados, WiFi en la banda de 2.4GHz y WiFi en la banda de 5GHz. Cuando se realizan experimentos con WiFi, éstos se pueden ver afectados por otras redes inalámbricas y por otras aplicaciones que usen esa banda de frecuencias. En nuestro caso, se han realizado los experimentos en presencia de otras redes WiFi de 2.4Ghz y una red de 5GHz. Como el objetivo es simular entornos reales, y la presencia de otras redes es un factor habitual que puede afectar a la QoS, no se ha aislado a los equipos contra estos fenómenos.

Los escenarios son:

- Escenario 1: este escenario consiste en conectar los dos equipos al mismo router directamente con cable, como se muestra en la Figura 3.1a, como escenario de referencia de un caso ideal con conexión directa con el servidor sin saltos inalámbricos.

- Escenario 2: en este escenario comprobamos las características de red que el router dd-wrt es capaz de medir. En este escenario, ilustrado en la Figura 3.1b, los resultados se ven claramente afectados por las limitadas capacidades computacionales del router.

- Escenario 3: como ya se ha mencionado, con la proliferación de los dispositivos tipo ODROID, es interesante investigar las capacidades de red que son capaces de aprovechar. Por ello, en este escenario se sustituye el PC1 del escenario 1 por una ODROID C2 como se muestra en la Figura 3.1c.

[5]Como en el caso de una red WiFi

Y los mismos escenarios pero sustituyendo un enlace cableado por un enlace WiFi:

- Escenario 4: en este escenario se conectan los dos equipos a través de una red WiFi, en la que el router TP-Link hace las veces de AP (*Access Point*, Punto de Acceso) y el router dd-wrt actúa de pasarela entre el PC2 y la red WiFi en la banda de 2.4GHz con un ancho de canal de 40MHz[6]. Este escenario se ilustra en la Figura 3.1d.

- Escenario 5: este escenario utiliza la misma topología que el escenario 4, pero ahora el equipo PC2 sólo se utiliza para establecer una sesión SSH al router dd-wrt, que es el que se utilizará para realizar las medidas. Esto evita mantener la sesión SSH por el enlace WiFi, que puede alterar las medidas realizadas.

- Escenario 6: en este escenario se miden las características de red que un usuario experimentaría si decide conectar su ODROID mediante el adaptador de red WiFi USB[7].

- Escenario 7: en este escenario se conecta el PC2 con el mismo adaptador de red USB del escenario anterior para ver el efecto que tiene sobre las métricas en este equipo.

- Escenario 8: en este escenario se conecta el PC2 con el PC1 manteniendo la topología de la Figura 3.1d pero con un enlace WiFi en la banda de 5GHz con un ancho de canal de 80MHz[8].

- Escenario 9: en este escenario se mantiene la topología anterior pero midiendo desde dd-wrt.

En todos los escenarios anteriormente descritos, se ha mantenido fijo el PC1 y el router TP-Link, que hacen las veces de "servidor" en estos experimentos, al que se conectan diferentes clientes mediante distintos tipos de enlace. Por ello, de ahora en adelante se utilizará siempre "Bajada" para referirse al sentido desde el PC1 al otro equipo; y "Subida" para referirse al sentido contrario.

En las mediciones de todos los escenarios hay un solo cliente, y los distintos canales de la comunicación se encuentran completamente disponibles para él, lo que da cotas máximas de rendimiento que se podrían experimentar. Estos resultados sirven como medida inicial, antes de adquirir más dispositivos (lo que aumentaría el coste) con los que realizar medidas en paralelo y estudiar los fenómenos de competición de acceso al medio que los clientes experimentarán. Sin embargo, no podemos olvidar que se están realizando avances que mejoran las tecnologías de acceso WiFi, aumentando el ancho de banda disponible (utilizando bandas y protocolos de acceso al medio diferentes [39]), y se publican estudios sobre su impacto sobre los protocolos de transporte [40], por lo que estos resultados no son definitivos, ya que dependen de muchos factores.

[6]con un índice MCS 22 en transmisión y MCS 23 en recepción
[7]con un índice MCS 7
[8]con un índice MCS de 8

3.2. Análisis de los resultados

Tabla 3.2: Medidas de ancho de banda en Mbits/s.

	TCP						UDP					
	Bajada			Subida			Bajada			Subida		
	$\hat{\mu}$	$\pm$	$\hat{\sigma}$	$\hat{\mu}$	$\pm$	$\hat{\sigma}$	$\hat{\mu}$	$\pm$	$\hat{\sigma}$	$\hat{\mu}$	$\pm$	$\hat{\sigma}$
Escenario 1	935.24	$\pm$	7.99	930.74	$\pm$	11.25	953.17	$\pm$	11.48	947.22	$\pm$	11.40
Escenario 2	384.80	$\pm$	7.98	179.72	$\pm$	1.99	10.15	$\pm$	0.33	269.83	$\pm$	8.40
Escenario 3	934.25	$\pm$	7.33	941.32	$\pm$	7.03	943.30	$\pm$	40.15	19.05	$\pm$	0.40
Escenario 4	164.82	$\pm$	9.81	111.77	$\pm$	10.20	53.47	$\pm$	1.56	44.45	$\pm$	1.46
Escenario 5	177.91	$\pm$	13.94	77.21	$\pm$	28.78	8.32	$\pm$	0.28	96.98	$\pm$	17.38
Escenario 6	96.63	$\pm$	9.56	52.93	$\pm$	18.82	45.00	$\pm$	4.64	62.00	$\pm$	17.06
Escenario 7	33.79	$\pm$	7.08	9.72	$\pm$	6.68	42.48	$\pm$	2.21	37.97	$\pm$	7.84
Escenario 8	274.93	$\pm$	12.81	213.44	$\pm$	12.44	246.79	$\pm$	7.80	42.93	$\pm$	1.27
Escenario 9	434.87	$\pm$	25.82	170.93	$\pm$	6.76	8.61	$\pm$	0.29	208.71	$\pm$	6.41

En la Tabla 3.2 se muestran las medidas de ancho de banda para cada escenario. En el escenario 2 se aprecia que `iperf3` ejecutado en el router dd-wrt es capaz de procesar el tráfico TCP el doble de rápido de lo que es capaz de generarlo. En bajada no es capaz de procesar más de 10Mb/s de tráfico UDP. Esto se debe a un *bug* conocido de `iperf3` que parece afectar a la versión 3.1.X[9] [10] que justifica que en el resto de escenarios inalámbricos (5 y 9) solo sea capaz de medir hasta 8Mb/s en bajada. Destaca la abultada reducción de ancho de banda cuando se utiliza UDP al medir con la ODROID. Este resultado fue muy similar en todas nuestras pruebas y se tuvo especial cuidado en que no hubiese fragmentación IP, lo que indica que en subida `iperf3` está generando el tráfico UDP de forma poco eficiente, problema probablemente relacionado con el *bug* anteriormente mencionado.

Comparando el escenario 6 con el 7 se conjetura que el ancho de banda medido con el adaptador Edimax depende de cómo se realicen las interconexiones con el USB y cómo lo gestiona el sistema operativo, ya que, a pesar de disponer a priori el PC2 de muchos más recursos computacionales, el ancho de banda percibido por éste es muy inferior al medido por la ODROID con exactamente la misma antena. Con estos resultados se puede observar que independientemente de la tecnología de acceso inalámbrico, el ancho de banda máximo percibido por un usuario se verá notablemente reducido frente al ancho de banda en bajada utilizando TCP. Comparando el escenario 8 frente al 4, se ve un aumento de más del 50 % en todas las medidas con UDP (salvo en subida). Se podría esperar un aumento mayor del ancho de banda medido en este experimento, pero debido a pérdidas esporádicas, TCP no es capaz de alcanzar mayor *throughput*, ya que el enlace en la banda de 5GHz es capaz de alcanzar tasas de hasta 440Mb/s.

[9] `https://github.com/esnet/iperf/issues/234`
[10] `https://github.com/esnet/iperf/issues/296`

Tabla 3.3: Medidas de RTT en milisegundos.

	Bajada			Subida		
	$\hat{\mu}$	$\pm$	$\hat{\sigma}$	$\hat{\mu}$	$\pm$	$\hat{\sigma}$
Escenario 1	0.221	$\pm$	0.069	0.192	$\pm$	0.044
Escenario 2	0.204	$\pm$	0.012	0.317	$\pm$	0.065
Escenario 3	0.600	$\pm$	0.038	0.629	$\pm$	0.029
Escenario 4	5.491	$\pm$	10.939	3.042	$\pm$	3.305
Escenario 5	3.318	$\pm$	4.535	2.987	$\pm$	4.281
Escenario 6	75.281	$\pm$	90.705	17.347	$\pm$	90.427
Escenario 7	16.045	$\pm$	75.715	92.718	$\pm$	302.146
Escenario 8	1.098	$\pm$	0.449	1.059	$\pm$	0.474
Escenario 9	0.926	$\pm$	0.063	1.029	$\pm$	0.292

Tabla 3.4: Medidas de *jitter* en milisegundos según `iperf3`.

	Bajada			Subida		
	$\hat{\mu}$	$\pm$	$\hat{\sigma}$	$\hat{\mu}$	$\pm$	$\hat{\sigma}$
Escenario 1	0.113	$\pm$	0.012	0.109	$\pm$	0.012
Escenario 2	0.018	$\pm$	0.007	0.004	$\pm$	0.001
Escenario 3	0.013	$\pm$	0.004	0.016	$\pm$	0.002
Escenario 4	0.020	$\pm$	0.016	0.056	$\pm$	0.130
Escenario 5	0.035	$\pm$	0.033	0.202	$\pm$	0.180
Escenario 6	0.238	$\pm$	0.156	0.554	$\pm$	1.112
Escenario 7	0.114	$\pm$	0.052	3.293	$\pm$	7.383
Escenario 8	0.036	$\pm$	0.005	0.038	$\pm$	0.012
Escenario 9	0.023	$\pm$	0.005	0.094	$\pm$	0.021

Tabla 3.5: Medidas de *jitter* en milisegundos a partir de `ping`.

	Bajada				Subida			
	$\hat{\mu}$	$\hat{\mu}_c$	$\hat{\sigma}$	$\hat{\sigma}_c$	$\hat{\mu}$	$\hat{\mu}_c$	$\hat{\sigma}$	$\hat{\sigma}_c$
Escenario 1	0.035	0.024	0.065	0.030	0.051	0.045	0.038	0.049
Escenario 2	0.013	0.008	0.012	0.014	0.071	0.057	0.074	0.052
Escenario 3	0.035	0.024	0.031	0.046	0.023	0.012	0.026	0.027
Escenario 4*	5.339	1.255	11.495	2.943	2.386	0.560	4.100	2.360
Escenario 5	3.624	1.580	5.738	4.720	2.627	0.352	5.307	2.090
Escenario 6	64.866	22.500	114.916	57.850	21.052	1.450	127.741	3.910
Escenario 7	15.488	0.840	75.423	2.920	82.314	0.390	266.506	2.345
Escenario 8	0.332	0.142	0.466	0.398	0.285	0.038	0.614	0.228
Escenario 9	0.383	0.028	0.806	0.392	0.200	0.033	0.379	0.230

En la Tabla 3.3 se muestran los resultados de medidas de latencia obtenidos utilizando `ping`. Se han marcado con un asterisco las medidas del escenario 4 ya que, de manera sostenida, de cada 100 sondas de `ping` se experimentaba un 10 % de pérdidas, de modo que los resultados mostrados son solo para las respuestas que llegaron.

En las Tabla 3.4 se muestran el `jitter` calculado por `iperf3` frente al calculado utilizando la Ecuación (2.4) con los resultados de `ping` en la Tabla 3.5. Se muestra tanto la media como la mediana $\hat{\mu}_c$, y tanto la desviación típica como el rango intercuartílico $\hat{\sigma}_c$ para ver el efecto de utilizar medidas robustas frente a valores atípicos. Estos resultados indican una gran dispersión de los valores, la presencia de valores atípicos (*outliers*) como es el caso de los escenarios 4 a 6 y de pérdidas ocasionales en el enlace.

Además, los resultados tan dispares entre las medidas de `iperf3` y las calculadas con `ping` no son contradictorios, ya que miden situaciones diferentes. `iperf3` envía todos los paquetes de manera que en cuanto se termina de transmitir un paquete se comienza inmediatamente a transmitir el siguiente (*back-to-back*). Es decir, el *jitter* en el caso de estar transmitiendo de manera sostenida y sin que el otro extremo transmita nada. Sin embargo, el *jitter* medido por `ping` se corresponde con la variabilidad en la latencia que sufren paquetes enviados periódicamente, lo que significa que aplicaciones que transmitan de manera periódica y determinista pero no sostenida sufrirán de mucha mayor variabilidad en la latencia percibida. Por otro lado, es interesante ver que el `jitter` en los escenarios inalámbricos es siempre superior en subida frente a la bajada según `iperf3` aunque según la Tabla 3.5 la situación es la contraria.

3.3. Conclusiones

En este capítulo se han explorado varios escenarios domésticos. Se han realizado medidas con equipamiento de gama baja-media, y se han medido las prestaciones de red que éstos ofrecen utilizando los principales indicadores de calidad.

Se ha comprobado que el ancho de banda que un dispositivo puede medir varía en gran medida según la capacidad de cómputo del mismo y del mecanismo de acceso a la red WiFi que utilice. Pese a que los dispositivos embebidos de bajo coste tipo ODROID no aprovechan todo el ancho de banda de la red inalámbrica, son capaces de alcanzar hasta 100Mb/s con TCP en una red de 2.4GHz a pesar de utilizar un adaptador económico. No obstante, con un PC de sobremesa se puede medir anchos de banda muy superiores a los 100Mb/s con TCP en la banda de 2.4GHz y de 200Mb/s en la banda de 5GHz conectado a través de otro router que gestione la comunicación inalámbrica. De hecho, se obtienen resultados aún mejores midiendo desde el propio router.

Las medidas de latencia dependen de la forma de conectarse a la red, ya que se han obtenido resultados cercanos a 1ms en los escenarios con una red WiFi en la banda de 5GHz frente a la gran variabilidad en la banda de 2.4GHz. En el momento de realizar estos experimentos la banda de 2.4Ghz estaba mucho más cargada que la de 5Ghz.

Por último, incluso con una utilización mínima del canal y en entornos controlados, los enlaces inalámbricos presentan pérdidas esporádicas, *jitter* elevado y un ancho de banda limitado, lo que puede afectar gravemente a la calidad de experiencia del usuario.

Estos resultados muestran la importancia capital de considerar estos retos a la hora de desplegar una red inalámbrica para implantar aplicaciones con requisitos de latencia y de variabilidad acotadas.

4

ODROID como dispositivo de captura

Antes de poder realizar ningún análisis pasivo, es necesario disponer de algún mecanismo de captura con el que obtener los datos. En esta sección se evalúan las capacidades de una ODROID C2 como dispositivo de captura. No se ha experimentado con el router dd-WRT debido a que tiene muy poca capacidad de cómputo, y como se verá en los resultados el proceso de captura es muy exigente en términos de CPU. Se han probado dos motores de captura del estado del arte: `dumpcap` y `netsniff-ng`. Estos motores de captura se han probado en la ODROID, mientras se le mandaba tráfico sintético a distintas tasas, y se ha evaluado el rendimiento en términos de paquetes capturados, tanto sin escribir los datos a un dispositivo de almacenamiento como escribiéndolos.

4.1. Entorno experimental

En estas pruebas se han utilizado el PC1 y la ODROID C2 del Capítulo 3, conectados mediante un cable gigabit ethernet. La ODROID, que hizo las veces del motor de captura, tiene como dispositivo de almacenamiento una memoria eMMC de 32GB[1].

Se plantean dos escenarios con los que evaluar las capacidades de la ODROID como dispositivo de captura:

- Escenario 1: el motor de captura escribe a `/dev/null`.
- Escenario 2: el motor de captura escribe a un fichero pcap.

[1] `http://www.hardkernel.com/main/products/prdt_info.php?g_code=G145622510341`

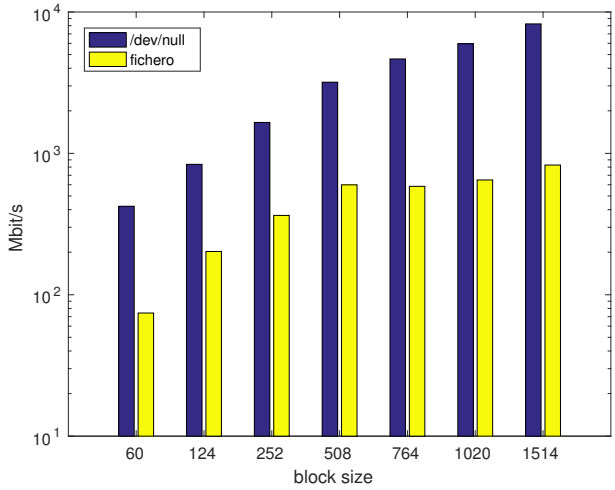


Figura 4.1: Velocidad de escritura de `dd` en la ODROID C2.

En la Figura 4.1 se muestra la máxima velocidad a la que una aplicación puede escribir en la memoria eMMC y en `/dev/null`. En la figura se muestra la media de 20 ejecuciones de dd en las que se escriben unos 640MB con distintos tamaños de *block size*, lo que simula una aplicación que esté escribiendo paquete a paquete a máxima tasa.

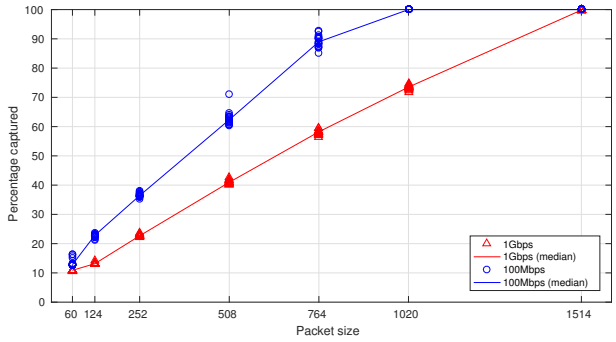
```
count=$((640*1024*1024/$bs))
for i in $(seq 1 20); do
        taskset -c 3 dd if=/dev/zero of=file bs=$bs count=$count
done
```

Código 4.1: Prueba realizada para medir el rendimiento de la memoria eMMC.

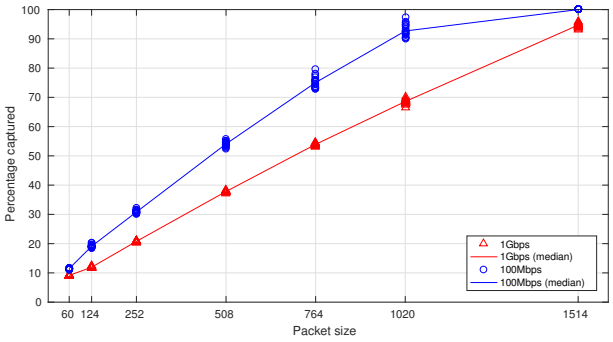
Se observa que con paquetes de tamaño mínimo de 64 bytes, la aplicación no podrá escribir todos los paquetes, ni siquiera con tráfico a 100Mbit/s. Además, solo con paquetes de tamaño máximo se puede alcanzar una tasa lo bastante cercana a Gb/s. Por tanto, si se desea escribir los paquetes, vamos a estar seriamente limitados por el dispositivo de almacenamiento, tanto por su capacidad como por su velocidad de escritura.

4.2. Resultados experimentales

En cada escenario se ha ejecutado tanto con `dumpcap` como `netsniff-ng` como motor de captura. Para cada prueba, se ha enviado tráfico sintético con un tamaño fijo generado con `trafgen`. Cada prueba ha durado unos 10 segundos, y se ha repetido 20 veces con tamaños de paquete 64, 128, 256, 512, 768, 1024 y 1518 (contando con CRC) cada una. Cuando a trafgen se le indica la tasa a la que debe transmitir, solo utiliza un *core* de la CPU de la máquina emisora, mientras que para transmitir a máxima tasa utiliza todos los *cores* de la máquina generadora del tráfico. Se ha comprobado que utilizando una sola cola de transmisión se obtiene mejor rendimiento que múltiples colas. De este modo, se ha fijado con `ethtool` una sola cola de transmisión en el PC1 para estas pruebas.

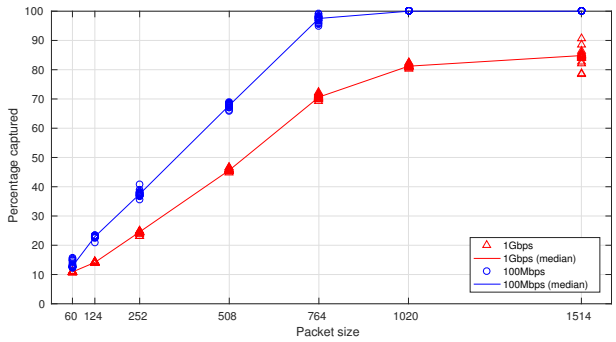


(a) Rendimiento de `dumpcap`

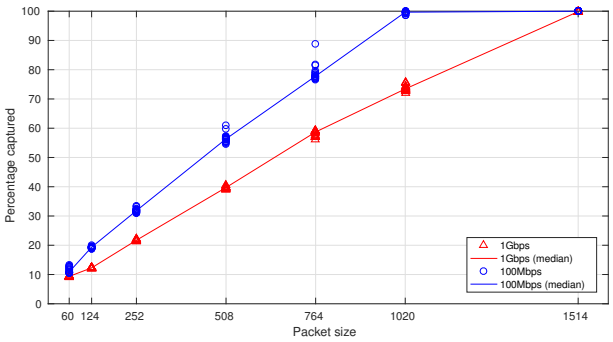


(b) Rendimiento de `netsniff-ng`

Figura 4.2: Comparativa de rendimiento de un motor one-copy frente a un zero-copy escribiendo a `/dev/null`.



(a) Rendimiento de `dumpcap`



(b) Rendimiento de `netsniff-ng`

Figura 4.3: Comparativa de rendimiento de un motor one-copy frente a un zero-copy escribiendo a la `eMMC`.

Además se ha comprobado que el rendimiento disminuye en un 5 ó 10 % si el tráfico tiene como dirección destino la IP asignada a la ODROID y no se ha lanzado alguna aplicación que pueda recibir ese tráfico, debido a que el *kernel* genera paquetes ICMP "Port Unreachable", lo que supone un sobrecoste no despreciable para el ya saturado *kernel*.

En la Figura 4.2 se muestra el rendimiento obtenido con cada motor de captura escribiendo a `/dev/null` para cada tamaño de paquete. Como se puede observar, a 100Mbit/s se obtiene un rendimiento muy pobre con paquetes con tamaño inferior a 768 bytes en el caso de `dumpcap`, que es entre un 5 y un 20 % superior a `netsniff-ng`. En todo caso, se ve que el rendimiento de ambos motores escala de manera lineal (hasta que satura con el tamaño máximo) a medida que aumentamos el tamaño de paquete. Este resultado es un poco sorprendente debido a que la interfaz es gigabit ethernet, pero tomando en cuenta el coste del dispositivo, tampoco se podría esperar que el rendimiento fuese a ser necesariamente parejo a un equipo con un coste 15 ó 20 veces superior.

Con tasa de 1Gb/s el rendimiento de `dumpcap` es superior al de `netsniff-ng` para todos los tamaños de paquete, y es llamativo que `netsniff-ng` no sea capaz de capturar el 100 % de los paquetes con tamaño máximo. Durante estas pruebas se ha visto que de

manera consistente el *core* 0 de la ODROID se mantiene con una utilización de 100 % durante cada prueba. Cabe destacar que el motor de captura siempre se ejecutó con la afinidad fijada al *core* 3[2], el resto de *cores* se mantenían desocupados, y que no parece que hubiese ningún proceso de usuario o de `root` cargando el *core* 0. Esto apunta a que el hilo de recepción del *kernel* de Linux está limitando el rendimiento de la ODROID como motor de captura.

La interfaz de red de la ODROID está muy limitada, y con `ethtool` no se puede modificar el número de colas de recepción ni muchos de los parámetros. Se probó a aumentar el tamaño de los *bufferes* de recepción del *kernel*, pero no se logró ninguna mejora aparente en el porcentaje de paquetes capturados. Estas limitaciones suponen que no hay ninguna optimización sencilla de la configuración que se pueda hacer para mejorar el rendimiento de manera notable.

En la Figura 4.3 se muestra el rendimiento obtenido con cada motor de captura escribiendo los paquetes a un fichero pcap. Cabe destacar el sorprendente aumento del rendimiento, de hasta un 5 %, cuando se escribe a fichero en lugar de `/dev/null`. Esto apunta a que al *kernel* le supone un mayor coste leer y descartar los datos del dispositivo `/dev/null` que copiarlos en el *buffer* correspondiente para que sean escritos en la tarjeta `eMMC`.

En cualquier caso, el rendimiento sigue siendo bajo para paquetes con tamaño inferior a 1024 bytes, y tiene como consecuencia que una ODROID C2 no es un dispositivo viable como motor de captura, salvo para redes en las que la tasa de trasmisión sea baja y el tamaño medio de paquete sea cercano o superior a 1024 bytes.

4.3. Conclusiones y observaciones finales

En este capítulo se ha explorado la viabilidad de dispositivos tipo ODROID C2 como motores de captura. Se ha comprobado que en las tasas habituales en una red doméstica o de oficina pequeña no es un dispositivo adecuado para capturar el tráfico y/o almacenarlo con ninguno de los dos motores de captura probados.

Además, se ha tratado de realizar el mismo estudio con una ODROID XU4. A priori, al poder conectar dispositivos de almacenamiento externos a través del USB 3.0, se podría utilizar como un dispositivo de captura económico que escribiese a un disco duro conectado a través del USB 3.0. Sin embargo, durante las pruebas no se han podido obtener resultados, debido a que durante la captura el hilo de recepción del *kernel* consumía toda la memoria de la ODROID XU4, lo que hacía que se quedara bloqueada y terminase por reiniciarse.

Por tanto, salvo con una tasa de paquetes y de ancho de banda muy bajas, estas ODROID no parecen adecuadas para ser utilizados como dispositivos de captura económicos.

[2]La ODROID tiene un procesador Cortex A53(ARMv8) quad core a 1.5Ghz

5

Análisis de algoritmos de detección de retransmisiones en flujos TCP

Para analizar los principales indicadores de la calidad de manera pasiva, en principio se pueden desarrollar algoritmos todo lo potentes que sea necesario para obtener una respuesta precisa. Sin embargo, cuando se desea analizar en tiempo real los paquetes que viajan por una red de alta tasa, esto impone unas restricciones muy estrictas sobre cuánto tiempo se puede invertir por cada paquete. Además, un factor importante a controlar es el coste de la máquina necesaria para ejecutar dichos algoritmos.

Una importante métrica indicadora de la calidad (KPI) es estimar las pérdidas de paquetes que están sufriendo los flujos TCP. Las retransmisiones de paquetes en un flujo TCP están altamente correlacionadas con las pérdidas y el desorden en los paquetes, pero son costosas de calcular. Por ello es interesante encontrar algoritmos que requieran pocos recursos que permitan obtener una estimación de las mismas. Esto da pie a que puedan ser implementados en dispositivos embebidos, o incluso en los propios routers de Internet.

En la práctica no tiene por qué ser tan importante saber con total exactitud el número de retransmisiones que tiene un flujo si éstas son muy costosas de contabilizar. Por ello, en este capítulo se presentan algoritmos heurísticos con los que calcular de manera aproximada el número de retransmisiones que tuvo un flujo TCP. Además, debido a que las tasas de retransmisiones habituales en Internet son bajas [41,42], se evalúa el rendimiento de los algoritmos para clasificar los flujos [18] en patológicos, si tuvieron al menos 100 paquetes entre sus dos sentidos y sufrieron más de un 5 % de retransmisiones en alguno de sus dos sentidos, y en no patológicos en otro caso. Adicionalmente, se presentan modelos analíticos con los que estimar la precisión y los recursos necesarios para ejecutar dichos algoritmos. Finalmente, se evalúan empíricamente estos modelos con trazas con tráfico real capturado en entornos empresariales y se extraen algunas conclusiones.

5.1. Algoritmos de detección de retransmisiones en flujos TCP

En el resto de esta sección utilizaremos la notación definida a continuación. Supongamos que tenemos una sesión TCP entre un *host* con dirección IP A, el cual envió el primer SYN, y el *host* con dirección IP B. De ahora en adelante se llamará *Cliente* al *host* que envió el primer SYN, y *Servidor* al otro *host*. Además, se utilizará la notación dada en Tabla 5.1 para el paquete n-ésimo del flujo TCP que va del *host* A hacia el B, y que se define de manera análoga para los paquetes que pertenecen al flujo que va del *host* B al A.

Tabla 5.1: Notación.

N	Número de paquetes en el flujo.
$l_n^{A\to B}$	Longitud de la carga (*payload*) TCP.[1]
$s_n^{A\to B}$	Número de secuencia del segmento TCP.
$ACK_n^{A\to B}$	Valor de ACK del segmento TCP.
Δ_n	Intervalo cubierto por el segmento n.

Se define el número de secuencia $s_n^{A\to B}$ tal que $\forall n \in \mathbb{N}, n \leq N, \exists k < n$:

$$s_n^{A\to B} = s_k^{A\to B} + l_k^{A\to B} \tag{5.1}$$

para algún valor inicial dado de $s_0^{A\to B}$. Nótese que si el paquete n-ésimo no contiene datos retransmitidos, entonces $k = n - 1$.

Se define el intervalo cerrado de números de secuencia, por abreviar *intervalo*, cubierto por el segmento en el paquete n del siguiente modo

$$\Delta_n = [s_n^{A\to B}, s_n^{A\to B} + l_n^{A\to B} - 1] \tag{5.2}$$

y se denotará el ínfimo de Δ_n con Δ_n^l, y su supremo con Δ_n^r.

5.1.1. Algoritmo completo de detección de retransmisiones TCP

Se considera que el paquete m-ésimo contiene datos retransmitidos si y solo si existe un $n < m$ tal que $\Delta_m \cap \Delta_n \neq \emptyset, n = 1, \ldots, N$, suponiendo que no haya tráfico duplicado. Se asume que el motor de captura es capaz de eliminar paquetes duplicados que puedan ser debidos, por ejemplo, al SPAN [19], como hace HPCAP [12].

Por tanto, para detectar si el paquete n-ésimo contiene datos retransmitidos es necesario mantener todos los intervalos anteriores Δ_m con $m < n$. Es decir, hay que seleccionar una estructura de datos que permita buscar de manera eficiente intervalos, y

[1] Como las banderas SYN y FIN consumen un número de secuencia, $l_n^{A\to B}$ es igual a la longitud de los datos mas uno por cada una de esas banderas que estén activas.

que permita detectar la intersección de un intervalo dado con cualquier otro que ya esté en dicha estructura. Además, se tiene que mantener una de estas estructuras por cada flujo. Esta estructura conceptualmente es una lista de intervalos[2], y por ello se denomina "lista de intervalos". A la hora de implementarla, puede ser interesante poder aplicar algoritmos de búsqueda, y éstos suelen requerir que se pueda definir una relación de orden entre los elementos de la estructura.

Para definir un orden entre los intervalos, se define la siguiente función de comparación

$$\Delta_i < \Delta_j \iff \Delta_i^l < \Delta_j^l \text{ or } (\Delta_i^l = \Delta_j^l \text{ and } \Delta_i^r \leq \Delta_j^r) \tag{5.3}$$

Si $\Delta_i < \Delta_j$ y $\Delta_i \cap \Delta_j \neq \emptyset$ éstos intervalos se pueden combinar en uno solo

$$\Delta' = [\Delta_i^l, \text{máx}(\Delta_i^r, \Delta_j^r)] \tag{5.4}$$

para mantener esta lista lo más compacta posible. De este modo, la lista se puede implementar como una lista enlazada ordenada, la cual se compactará siempre que sea posible para ahorrar memoria.

Sin embargo, durante un escenario real de monitorización hay que tener en cuenta que los paquetes pueden llegar desordenados o se puede perder un paquete durante la captura, los cuales pueden producir huecos en dicha lista. Por tanto, el algoritmo descrito hasta ahora no puede detectar paquetes que contengan datos retransmitidos si estos fueron enviados originalmente en paquetes que se perdieron antes del punto de captura (o durante la captura). Es más, pueden producirse huecos permanentes si un paquete se pierde durante la captura pero el paquete original llega al *host* destinatario, de modo que nunca será retransmitido.

Asumiendo que no lleguen paquetes fuera de orden, una forma de solucionarlo es utilizando el valor de $ACK_n^{B \to A}$ para rellenar huecos en la lista del otro sentido, $A \to B$; ya que indica que todos los números de secuencia hasta ese valor han llegado al *host* B. De hecho, si se activan extensiones de TCP como *Selective ACKnowledment* (SACK) éstas se pueden usar igualmente para rellenar huecos en la lista del otro sentido. Otra extensión que se puede aprovechar es la de marcas de tiempos (*timestamps*) para poder detectar si los paquetes están llegando desordenados o si se están produciendo retransmisiones de paquetes perdidos durante la captura.

En definitiva, este proceso es complicado, y requiere tener en cuenta muchas excepciones y sutilezas para ser implementado correctamente. Además, no parece escalar a velocidades arbitrarias ya que es costoso y potencialmente puede requerir mucha memoria, lo cual lo limita, a su vez, debido a que cuando la tasa de paquetes es muy elevada la latencia de acceso a memoria comienza a ser un factor limitante de este algoritmo. Por tanto, es interesante explorar alternativas aproximadas que puedan ser implementadas en dispositivos embebidos, o que incluso puedan ser más adecuadas para ser desplegadas como una función virtual de red (NFV).

[2]aunque no necesariamente deba implementarse como una lista

5.1.2. Detección aproximada de retransmisiones TCP

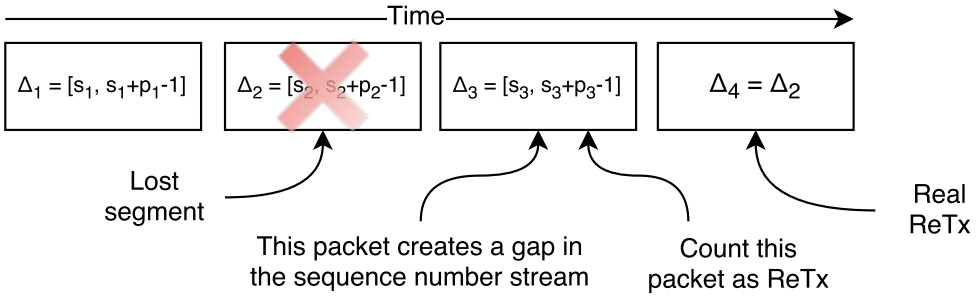


Figura 5.1: Algoritmo basado en contar paquetes que generan huecos.

En esta sección se van a repasar dos algoritmos heurísticos para detectar flujos con muchas retransmisiones presentados en [18], y se van a proponer versiones generalizadas de los mismos.

Las heurísticas se basan en la siguiente suposición:

> **Proposición 5.1.1.** *en una implementación típica de TCP, los paquetes se transmiten ordenados y cada segmento lleva un número de secuencia* $s_n^{A\to B}$ *que es igual al anterior más la carga que llevaba.*

Es decir que dado el paquete n-ésimo entonces

$$s_{n+1}^{A\to B} = s_n^{A\to B} + l_n^{A\to B} \tag{5.5}$$

si no hubo ninguna retransmisión, como se mostró en la Ecuación (5.1). De hecho, bajo esta suposición, solo los paquetes que contengan datos retransmitidos no cumplirán la Ecuación (5.1).

Por tanto, se proponía solo almacenar un único intervalo, Δ_l, por cada flujo TCP, y realizar una comparación por cada nuevo paquete para decidir si contenía datos retransmitidos o no. Se propusieron dos alternativas

- El primer algoritmo consiste en lo siguiente: dado el paquete n-ésimo, basta con contar aquellos paquetes tales que $\Delta_n^l > \Delta_l^l$. Estos paquetes crearían un hueco en el *buffer* del receptor, $s_{n+1}^{A\to B} > s_n^{A\to B} + l_n^{A\to B}$, y bajo la suposición 5.1.1 esto indica que dicho hueco se ha producido por una pérdida, que deberá ser retransmitida más tarde.

  En la Figura 5.1 se ilustra este fenómeno. Primero se pierde el segundo paquete antes de la captura, y por tanto el sistema de monitorización no lo verá. Entonces al llegar el tercer paquete, se detecta que éste causa un hueco y se cuenta como una retransmisión. Finalmente, el cuarto paquete es la verdadera retransmisión, pero no se cuenta como tal. En la práctica, la retransmisión no tendría por qué ser el cuarto paquete, podría ocurrir muchos paquetes después, pero a ojos del algoritmo es indiferente. En el resto del documento nos referiremos a este algoritmo como *algoritmo basado en huecos.*

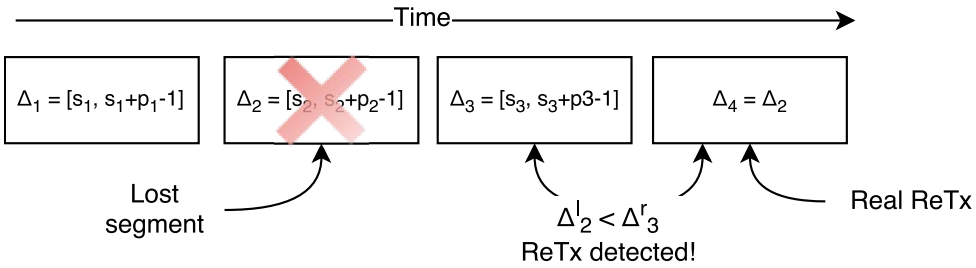


Figura 5.2: Algoritmo basado en contar paquetes desordenados.

- El segundo algoritmo consiste en contar aquellos paquetes cuyo número de secuencia fuese menor o igual al esperado $(\Delta_l^r + 1)$, *i.e.*, $s_{n+1}^{A\to B} < s_n^{A\to B} + l_n^{A\to B}$. Bajo la suposición 5.1.1 estos paquetes son retransmisiones.

  En la Figura 5.2 se ilustra este algoritmo utilizando la misma situación que en el caso anterior. De hecho, en este caso, tras recibir el tercer paquete, el cuarto paquete que tiene como número de secuencia $s_2^{A\to B}$ es menor que el esperado, y por tanto se cuenta como una retransmisión. En general, no tendría por qué ocurrir que el paquete que se cuente como retransmisión sea la retransmisión real, como es en este caso. En el resto del documento nos referiremos a este algoritmo como *algoritmo basado en desorden*.

La consecuencia evidente de utilizar estos algoritmos es que son muy eficientes y que tienen unos requisitos de memoria óptimos, desde el punto de vista del seguimiento de números de secuencia.

Como versión generalizada de estos algoritmos, se propone que se comporten de manera similar al algoritmo completo, pero con una cota superior del número máximo de intervalos que pueden almacenar. Es decir, se evaluará si el nuevo intervalo interseca con alguno de los almacenados, se insertará de la manera habitual y se tratará de compactar la lista. En caso de que haya más intervalos del límite permitido, los más antiguos serán descartados. Además, se proponen las siguientes reglas adicionales para contar retransmisiones:

- En el caso del *algoritmo basado en huecos*, se aplicará la heurística a cada paquete que se inserte en la lista si genera un nuevo hueco en la misma.
- En el caso del *algoritmo basado en desorden*, se contarán como retransmisiones aquellos paquetes que se deban descartar cuando la lista haya superado la cantidad permitida de intervalos.

De este modo, si solo se permite llevar un intervalo por flujo, estas versiones generalizadas se comportan de igual modo que las heurísticas originales.

5.2. Modelado del error

Para modelar el error de los algoritmos descritos en la sección anterior se propone un modelo analítico para la probabilidad de clasificar de manera incorrecta un flujo como patológico en términos del número de retransmisiones. En Tabla 5.2 se recogen los parámetros de este modelo.

Tabla 5.2: Parámetros del modelo del error.

p	$\mathbb{P}\{$paquete clasificado como retransmisión incorrectamente$\}$
q	$\mathbb{P}\{$retransmisión no detectada$\}$
$1-p-q$	$\mathbb{P}\{$paquete clasificado correctamente$\}$
N	Número máximo de paquetes un flujo puede tener

Dado un flujo con N paquetes, se define $\mathcal{E}_N$ como el error total de clasificación de dicho flujo:

$$\mathcal{E}_N = \#\{\text{ReTx según referencia}\} - \#\{\text{ReTx según algoritmo}\} \tag{5.6}$$

es decir, $\mathcal{E}_N$ es la acumulación de errores de clasificación de cada uno de los paquetes del flujo.

De este modo, tenemos que p representa la probabilidad de que $\mathcal{E}_1 = -1$, q es la probabilidad de $\mathcal{E}_1 = +1$, y $1-p-q$ es la probabilidad de $\mathcal{E}_1 = 0$. Asumiendo que la probabilidad de clasificación de cada paquete es independiente, podemos modelar $\mathcal{E}_N$ como una cadena de Markov finita e irreducible, donde cada estado representa la suma acumulada de cada uno de los errores de clasificación de cada paquete.

De hecho, p y q son las probabilidades de tránsito de cada uno de los estados al siguiente o al anterior, y $1-p-q$ la probabilidad de mantenerse en el mismo estado. Esto se ilustra en la Figura 5.3.

Para esta cadena de Markov tenemos $2N+1$ estados, debido a que cabe la posibilidad tanto de cometer el mismo tipo de error en todos los paquetes, como cualquier estado intermedio entre los dos extremos.

Por tanto, tenemos una matriz de transición $(2N+1) \times (2N+1)$ con la siguiente estructura tridiagonal:

$$\mathcal{P} = \begin{pmatrix} 1-q & q & 0 & \dots & & & \\ & & & \ddots & & & \\ \dots & 0 & p & 1-p-q & q & 0 & \dots \\ & & & \ddots & & & \\ & & & \dots & 0 & p & 1-p \end{pmatrix} \tag{5.7}$$

Esta estructura se debe a que por construcción del modelo, dado un estado solo se puede saltar al siguiente, al anterior o mantenerse en el mismo estado; ya que se considera la clasificación de cada paquete como un proceso iterativo.

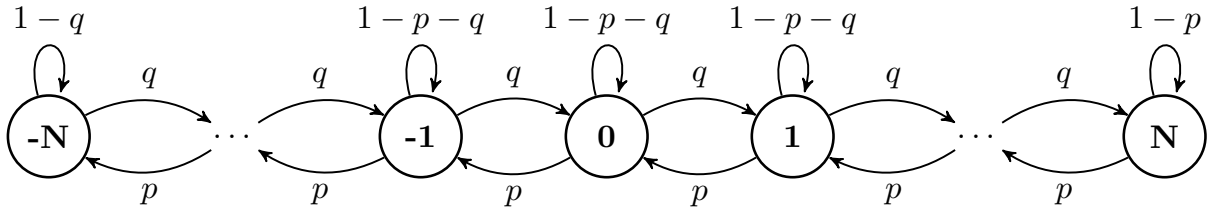


Figura 5.3: Cadena de Markov de los errores acumulados.

Cabe destacar que el error de clasificación de un flujo, $\mathcal{E}_N \in [-N, N]$, es una variable aleatoria discreta cuya función de masa de probabilidad se calcula mediante $\lambda \mathcal{P}^N$, donde

$$\lambda = (\overbrace{0, \ldots, 0}^{N \text{ veces}}, 1, \overbrace{0, \ldots, 0}^{N \text{ veces}}) \tag{5.8}$$

es la distribución inicial. Este cómputo se puede realizar directamente si la matriz es pequeña o, si las dimensiones son grandes, se puede realizar utilizando el método de la potencia [43], con la fórmula:

$$\pi_0 = \lambda, \qquad \pi_k = \pi_{k-1}\mathcal{P} \qquad k = 1, \ldots, N \tag{5.9}$$

Aprovechando que $\mathcal{P}$ es *sparse*, la Ecuación (5.9) puede ahorrar tanto memoria como número de operaciones necesarias para calcular la distribución del error condicionada al tamaño del flujo.

Para poder calcular la esperanza del error de clasificación de un flujo independientemente del número concreto de paquetes que tenga N, tenemos que incorporar la distribución del número de paquetes en un flujo a nuestro cálculo. Se ha observado que el número de paquetes en un flujo en Internet suele seguir una distribución de Zipf [44,45], o al menos, en la cola de la distribución. Por tanto, podemos tomar la distribución de masa de probabilidad del número de paquetes en un flujo del siguiente modo [46]:

$$\mathbb{P}(X = x) \approx K x^{-\alpha}, \quad K > 0,\ x \geq x_{min},\ \alpha > 0 \tag{5.10}$$

donde K es la constante de normalización, x_{min} es un cierto valor a partir del cual tratamos de ajustar la distribución, y típicamente $2 < \alpha < 3$.

Por tanto, podemos finalmente expresar la esperanza del error de clasificación $\mathbb{E}(|\mathcal{E}_N|)$ del siguiente modo

$$\begin{aligned}\mathbb{E}(|\mathcal{E}_{N_{max}}|) &= \sum_{\varepsilon=-N_{max}}^{N_{max}} |\varepsilon| P(\mathcal{E} = \varepsilon) = \sum_{\varepsilon=-N_{max}}^{N_{max}} |\varepsilon| \sum_{n=x_{min}}^{N_{max}} P(\mathcal{E} = \varepsilon | N = n) P(N = n) = \\ &= \sum_{n=x_{min}}^{N_{max}} \sum_{j=1}^{2n+1} |j - 1 - n| \left(\lambda \mathcal{P}^n\right)_k K\, n^{-\alpha}\end{aligned} \tag{5.11}$$

donde N_{max} es el supremo del número de paquetes que cualquier flujo de una traza puede tener y K es la constante de normalización de la distribución de los paquetes de la traza de modo que tenga una masa total igual a 1.

Por último, cabe destacar que por simetría, dada una tupla de probabilidades de error (p, q), la distribución del error $\mathcal{E}_N$ es simétrica a la obtenida con la tupla (q, p). Además, en simulaciones se ha observado que para un valor lo suficientemente grande de N, el error esperado tiende a una asíntota, que depende tanto de α, q y q.

5.3. Modelo del uso de memoria

La cantidad de uso de memoria es importante, tanto desde el punto de vista de los recursos de los que debe disponer el sistema para poder ejecutar una aplicación, como en términos de eficiencia, debido a que un menor uso de memoria por cada estructura de datos puede explotar las bondades de localidad espacial que aprovechan las cachés si el patrón de acceso a memoria lo permite. Además, el rendimiento también puede verse limitado por la latencia de las escrituras y lecturas a memoria. Por tanto, debido a que cuanto más rápidas son las memorias menos capacidad tienen, hay que encontrar un compromiso entre rendimiento y capacidad de la memoria que la aplicación puede utilizar.

El consumo de memoria vendrá en función del número de flujos concurrentes que se desea poder seguir en el tráfico analizado. Por tanto, para obtener una estimación de la cantidad de memoria necesaria, hay que analizar los recursos necesarios por cada flujo para poder extrapolarlo al sistema completo.

Se define

- λ_t, la tasa a la que llegan nuevas conexiones TCP al sistema de monitorización.
- W_t, el tiempo medio de estancia en el sistema. Este tiempo está relacionado con la duración de una sesión TCP y con los parámetros de recolección de basura del sistema a la hora de reciclar la memoria disponible.
- M, el número máximo de intervalos que un flujo TCP puede ocupar en memoria. En caso de necesitar más, hay que definir una política para descartar los intervalos sobrantes.
- I_t^M, el número esperado de intervalos en la lista de intervalos, que está acotado por M.
- L_t^M, el número total esperado de intervalos en todo el sistema cuando el número máximo por cada flujo es M.

Con las definiciones anteriores y siguiendo la Ley de Little [47], el número medio de intervalos en el sistema está dado por la Ecuación (5.12).

$$L_t^M = \lambda_t \cdot W_t \cdot I_t^M \tag{5.12}$$

Por tanto, se pueden comparar los distintos algoritmos en función de sus valores de M. En concreto, es interesante comparar los algoritmos propuestos en [18] frente a un

Tabla 5.3: Requisitos de memoria en MB.

# sesiones TCP concurrentes	6.4M	3.2M	1.6M	800K	400K	200K	100K
estado mínimo algoritmo completo	512	256	128	64	32	16	8
1 *intervalo* por flujo algoritmo completo	716.8	358.4	179.2	89.6	44.8	22.4	11.2
1.1 *intervalos* por flujo	747.52	373.76	186.88	93.44	46.72	23.36	11.68

valor cualquiera de M con la métrica dada en la Ecuación (5.13).

$$\Delta L_t^M = L_t^M - L_t^1 = \lambda_t \cdot W_t \cdot (I_t^M - I_t^1) \tag{5.13}$$

Cabe destacar que W_t es igual al tiempo medio de estancia en el sistema más el tiempo adicional que resida en memoria hasta que el flujo se exporte [48].

Se distinguen dos tipos de sesiones TCP en función del motivo por el que expiran:

- Banderas TCP: la sesión expira tras observar la banderas FIN o RST. Sin embargo, no puede expirar inmediatamente, porque puede haber todavía paquetes en tránsito por la red. Esto está contemplado en el propio protocolo TCP, donde en el RFC 1122 [49] se define un estado *TIME-WAIT* durante el cual un emisor TCP debe esperar a estos segmentos que siguen en tránsito, y se define que tiene una duración de dos veces el *Maximum Segment Lifetime* (MSL), con MSL definido con una duración arbitraria de 120 segundos.

- Inactividad: la sesión expira porque no se observaron paquetes durante un intervalo de tiempo dado. Esto es para evitar mantener en memoria registros de sesiones TCP que se han terminado pero no hay evidencia de esto en la traza (quizá por una pérdida de conectividad entre los *hosts*). Por tanto, se define como parámetro del sistema de monitorización el tiempo máximo de inactividad, *Inactivity Time-Out* (ITO), que se ha fijado de manera arbitraria a 15 minutos (900 segundos) durante los experimentos ya que se ha comprobado que es una cantidad suficiente para la mayoría de los flujos.

Por tanto, se define

$$W_t = D_t + c_1 \cdot (2 \cdot MSL) + c_2 \cdot ITO \tag{5.14}$$

con D_t la duración aleatoria de una sesión TCP y c_1,c_2 son las proporciones esperadas de flujos TCP expirados por banderas o por inactividad, respectivamente.

Por construcción, tenemos que c_1 y c_2 son probabilidades de conjuntos complementarios, por lo que $c_1 + c_2 = 1$. Por tanto, ambos están relacionados con el número de sesiones que expiraron por inactividad. Aumentar el valor de ITO reduce el valor de c_2 (aumentando de la misma manera c_1) y viceversa, ya que un valor mayor de ITO reduce la probabilidad de que un flujo expire por inactividad.

En la Tabla 5.3 se muestran estimaciones del uso de memoria necesario para el sistema con diferentes configuraciones. Para estimar la cantidad de memoria necesaria, se supone un mínimo de 64 bytes de información por cada sesión TCP. En ellos se incluye la 4 tupla (12 bytes), contadores de bytes, paquetes y retransmisiones (distinguiendo por sentido) (6×8 bytes), y una marca de tiempo de último acceso (4 bytes) para poder implementar el mecanismo de expiración por inactividad. Además, durante la evaluación empírica 5.4 se ha comprobado que en media una sesión requiere de 1.1 intervalos, *i.e.*, el sistema tiene en total solo un 10 % más intervalos que flujos concurrentes.

Para poder implementar el algoritmo completo, son necesarios al menos 24 bytes adicionales por cada intervalo (2 punteros y 2 enteros), y al menos se requieren 2 por cada sesión, llegando a un total de 112 bytes por cada flujo. En este caso, se requieren unos 750MB para poder mantener 6.4 millones de flujos concurrentes TCP.

Por otro lado, en el caso de reducir al mínimo el estado por cada sesión, podemos reducirlo hasta 80 bytes por sesión en el caso de utilizar solo un número de secuencia por cada flujo (2×2×4 bytes por cada sesión) de manera que la cantidad de memoria necesaria para mantener 6.4 millones de flujos concurrentes se reduciría a 512MB.

Por tanto, esta reducción de aproximadamente un 30 % de la memoria necesaria facilita la implementación de los algoritmos más simples en dispositivos embebidos como una FPGA [10] sin necesidad de utilizar memorias externas pero aún así pudiendo mantener el estado de muchas conexiones concurrentes.

5.4. Evaluación experimental

En esta sección se presenta una evaluación empírica de la precisión y de la cantidad de memoria necesaria para analizar tráfico de trazas tomadas en entornos empresariales.

5.4.1. Entorno experimental

Para la evaluación experimental se han utilizado tres conjuntos de datos (denominados A, B y C) de tráfico real *HyperText Transfer Protocol* (HTTP) capturado en entornos empresariales y cuyas características se resumen en la Tabla 5.4. Lo primero que se hizo fue comprobar que la distribución del tamaño de los flujos se podía ajustar por la Ecuación (5.10). En el Apéndice A se incluyen gráficas con las distribuciones empíricas del número de paquetes por flujo y el ajuste resultante. Los parámetros α y K utilizados se recogen en la Tabla 5.4.

Al tratar de realizar la evaluación frente a Tshark, se comprobó que ésta no iba a ser posible, debido a que Tshark primero carga la traza entera en memoria, y luego la analiza. De hecho. Tshark requería más memoria de la que ocupaban las traza en disco, debido a que durante el análisis necesita poder almacenar estado.

Tabla 5.4: Conjuntos de datos utilizados en las pruebas. (*) Número de sesiones TCP con al menos cien paquetes.

Traza	Tamaño (GB)	Paquetes (millones)	Tamaño medio de paquete	Sesiones TCP*	Flujos* con ReTx	% medio de paquetes fuera de orden	Cliente a Servidor		Servidor a Cliente	
							α	K	α	K
A	91	149	634B	98279	2972	0.15	2.14	13.00	1.92	6.68
B	120	211	754B	196580	705	$1{,}97 * 10^{-3}$	2.28	19.71	1.88	3.93
C	387	539	591B	725648	13721	$2{,}16 * 10^{-4}$	2.35	24.02	2.03	9.24

Por tanto, para realizar el análisis de precisión se hizo un muestreo simple estratificado de la traza C, ya que es la que tiene más retransmisiones. Se tomaron al azar 2500 sesiones TCP (5000 flujos) sin retransmisiones, y otras 2500 sesiones TCP (otros 5000 flujos) con retransmisiones en al menos uno de sus dos flujos, de los cuales, un 14 % sufrían de muchas retransmisiones. Estos flujos, con más de 5 millones de paquetes en total, se extrajeron en trazas individuales, y éstas se analizaron con Tshark paquete a paquete. Después, se procesó esa salida para construir registros de flujo similares a los que genera el programa desarrollado, para facilitar la comparación de la clasificación de flujos asegurando que tienen una la misma estructura e información.

5.4.2. Evaluación de la precisión frente al algoritmo completo

Lo primero que se hizo fue analizar la precisión de clasificación en flujos patológicos de los algoritmos frente al algoritmo completo. Como se vio en [18] con un solo número de secuencia el *algoritmo basado en desorden* tiene tasas de error inferiores al 1.5 %, frente al algoritmo *basado en huecos*, que tiene tasas de error muy elevadas. Al evaluar las versiones generalizadas de los algoritmos con $M \in \{1, 4, 10000\}$ números de secuencia[3] no se vieron cambios significativos a los mostrados en la Tabla 5.5.

Dados estos resultados, se decidió descartar el *algoritmo basado en huecos* y el resto de la evaluación se hizo solo con el *algoritmo basado en desorden*.

Tabla 5.5: Resultados experimentales frente al algoritmo completo con solo un número de secuencia por flujo.

Traza	Cliente a Servidor				Servidor a Cliente			
	FNR (%)		FPR (%)		FNR (%)		FPR (%)	
	desorden	*huecos*	*desorden*	*huecos*	*desorden*	*huecos*	*desorden*	*huecos*
A	0	62.56	0.69	17.18	0	96.50	1.41	0.91
B	0	88.89	0.02	0.01	0	96.82	0.09	0.01
C	0	14.29	0.01	13.19	0	99.88	0.02	0.01

[3]esto nos permite en la práctica almacenar todos los números de secuencia que sean necesarios, como haría el algoritmo completo

5.4.3. Evaluación de la precisión frente a Tshark

En las Tablas 5.6a a 5.6c se muestran las matrices de confusión de la clasificación de los paquetes con el algoritmo de desorden con distintos valores de M.

Según Tshark hay un total de 60621 retransmisiones, aunque hay un subconjunto de retransmisiones que no cuenta, debido a que Tshark considera una pequeña ventana de reordenación de 3ms que altera la clasificación de esos paquetes[4] de retransmisión a paquetes "fuera de orden" (*out-of-order packets*). Esto quiere decir que todo paquete que se considere una retransmisión pero haya ocurrido en menos de 3ms desde segmento con el número de secuencia más alto, se cambia la categoría de ese paquete a "fuera de orden". Estos paquetes representan un pequeño porcentaje de los paquetes que se clasifican como retransmitidos, y explica que haya un 9 % de falsos positivos en la clasificación del *algoritmo basado en desorden*.

Se han calculado algunas de las métricas clásicas de clasificación [50] como *Positive Predictive Value* (PPV), *True Positive Rate* (TPR), *True Negative Rate* (TNR) o *accuracy*, que se recogen en la Tabla 5.7. Se observa que al aumentar el número máximo de intervalos por flujo, M, mejora la precisión (PPV), pero se reduce ligeramente la sensibilidad (TPR), y apenas afecta a la especificidad (TNR) o al *accuracy*.

De hecho, utilizando el índice de Jaccard [50]

$$Jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{5.15}$$

que mide la similitud entre dos conjuntos, se ha comprobado que el conjunto de las retransmisiones detectadas por el algoritmo y por Tshark es de casi un 90 % en todos los casos.

Con estos resultados se ha podido estimar los valores de (p, q) del modelo del error dado en la Sección 5.2, y se ha obtenido que $p \approx 1{,}2 \times 10^{-3}$ y $q \approx 1{,}9 \times 10^{-4}$, lo que según el modelo se traduce en un error de clasificación inferior al 1.5 % por cada flujo.

Una vez realizado este análisis, se continuó evaluando la clasificación a nivel de flujos patológicos y no patológicos. Cabe destacar que incluso si los resultados a nivel de paquete tuviesen unas tasas de error muy elevadas, la clasificación a nivel de flujo podría ser

Tabla 5.6: Matrices de confusión del análisis paquete a paquete.

(a) Matriz de confusión para $M = 1$

$M = 1$		Predicción ReTx	Predicción No ReTx
Real	ReTx	59659	962
	No ReTx	6021	5114647

(b) Matriz de confusión para $M = 4$

$M = 4$		Predicción ReTx	Predicción No ReTx
Real	ReTx	59652	975
	No ReTx	5959	5114716

(c) Matriz de confusión para $M = 10^4$

$M = 10^4$		Predicción ReTx	Predicción No ReTx
Real	ReTx	59638	987
	No ReTx	5956	5114733

[4] `https://ask.wireshark.org/questions/27662/how-to-understand-out-of-order-tcp-segments/27664`

Tabla 5.7: Métricas del análisis paquete a paquete.

M	PPV (%)	TPR (%)	TNR (%)	Accuracy (%)
1	90.83	98.41	99.88	99.88
4	90.92	98.39	99.88	99.88
10000	90.92	98.37	99.88	99.89

correcta. El objetivo de los algoritmos no es clasificar los paquetes, sino obtener el número de retransmisiones del flujo, y por diseño, no tienen por qué clasificar correctamente los paquetes para obtener el número de retransmisiones, como se ilustró en la Sección 5.1.2.

En la Tabla 5.8 se recogen los resultados de clasificación de los flujos. De los 5000 flujos de Cliente a Servidor, todos fueron clasificados correctamente, y esto se ve reflejado en las tasas *False Negative Rate* (FNR) y *False Positive Rate* (FPR). En los flujos de Servidor a Cliente hay más falsos negativos en la clasificación de los paquetes, lo que hace que algunos flujos se clasifiquen a su vez de manera incorrecta como no patológicos. Además cabe destacar que tanto el algoritmo completo como el algoritmo de detección de retransmisiones por desorden con solo un número de secuencia tienen la misma tasa de error.

Tabla 5.8: Clasificación de los flujos frente a Tshark.

Algoritmo	Cliente a Servidor		Servidor a Cliente	
	FNR (%)	FPR (%)	FNR (%)	FPR (%)
Algoritmo completo	0	0	0	2.38
Algoritmo de desorden, $M = 1$	0	0	0	2.38
Algoritmo de desorden, $M = 4$	0	0	0	2.38
Algoritmo de desorden, $M = 10^4$	0	0	0	2.38

5.4.4. Evaluación del consumo de memoria

Para realizar esta evaluación se instrumentó la implementación de los algoritmos para que de manera periódica (1 segundo) imprimiesen estadísticas de utilización de recursos. De esta manera, se comprobó que salvo que haya mucho desorden en los paquetes de un mismo flujo (nótese que cómo se entremezclen paquetes de flujos diferentes no afectan en absoluto al uso de memoria de estos algoritmos) un número de secuencia es más que suficiente si la lista de intervalos se mantiene compactada. De hecho, el caso peor ocurría en la traza C, donde solo eran necesarios en media 1.1 intervalos por flujo.

En la Figura 5.6 se muestran las distribuciones del número medido de intervalos frente al predicho por el modelo en el caso de $M = 1$, tomando como W_t el tiempo medio de estancia en el sistema y acotándolo además por el percentil 10 y 90, para obtener cotas de caso peor y caso mejor. Además, se muestra el error relativo entre el modelo y el real, y se puede comprobar que se obtiene un error inferior al 20 % para el 90 % del tiempo.

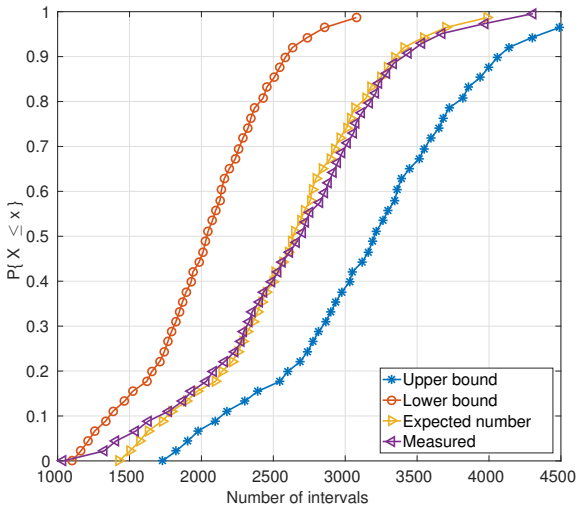


Figura 5.4: Valores distribucionales.

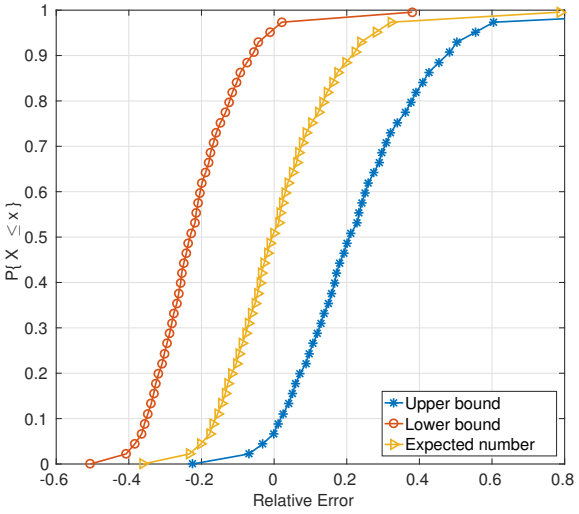


Figura 5.5: Error relativo.

Figura 5.6: Comportamiento del uso de memoria predicho por el modelo frente al medido durante los experimentos.

5.5. Conclusiones

En este capítulo se han revisado varios algoritmos de detección de retransmisiones dados en [18], se ha explorado su generalización a más de un intervalo por flujo y se han presentado y evaluado sendos modelos del error y de la cantidad de memoria requerida por estos algoritmos. Durante la evaluación se ha comprobado que la versión generalizada del algoritmo de detección de retransmisiones por desorden con solo un número de secuencia es suficiente para clasificar correctamente los flujos en patológicos o no patológicos. Estos resultados se resumen en la Tabla 5.9, donde se muestra una comparativa entre los sistemas de monitorización que se han usado en este trabajo.

Tabla 5.9: Comparativa de las herramientas evaluadas.

Sistema	Requisitos de memoria	Mantiene estado completo de la sesión	Detección de retransmisiones TCP	Adecuado para grandes volúmenes de datos
Tshark	O (paquetes)	✓	✓	✗
Algoritmo de desorden	O (flujos TCP)	✗	✓/	✓

6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

En este trabajo se han explorado las características de dispositivos ODROID y routers como dispositivos tanto de medida activa como pasiva. Desde el punto de vista de la medida activa, se han utilizado herramientas bien conocidas, `iperf3` y `ping`, y se ha comprobado que las medidas tomadas en medios compartidos son muy problemáticas y variables, debido a que son sensibles a muchos factores difíciles de controlar. Además, dependiendo de la calidad del hardware utilizado, se obtienen resultados muy dispares, y no parece probable que un usuario vaya a comprar una antena tan cara o más que una ODROID solo para obtener mejor calidad de servicio. Esto pone de nuevo de manifiesto la importancia de medir la calidad de servicio, ya que ciertas aplicaciones sensibles a estos parámetros no funcionarán si no se cumplen ciertos requisitos mínimos.

Desde el punto de vista de la medida pasiva, la ODROID C2 ha demostrado estar muy limitada al intentar utilizarla como dispositivo de captura de alto rendimiento, pero es una alternativa viable en redes con baja tasa y tamaño medio de paquete en torno a los 750 bytes, que parece que es hacia lo que tiende el tráfico web en Internet [12].

Finalmente, se ha continuado el trabajo iniciado en [18] de detección de retransmisiones en flujos TCP, planteando versiones generalizadas de los algoritmos. Se ha realizado una evaluación de la efectividad de los algoritmos frente a la herramienta de referencia, Tshark. De esta evaluación se ha concluido que el algoritmo de detección de desorden con solo un número de secuencia es igual de efectivo que el algoritmo completo. Además, se han presentado modelos analíticos del error que cometen los algoritmos y la cantidad de memoria que requieren, que han probado ser válidos con los conjuntos de datos que se han usado.

6.2. Trabajo Futuro

A continuación se comentan algunas líneas de trabajo futuro:

- Evaluar el efecto de la concurrencia de múltiples usuarios sobre las medidas activas, en especial cuando están todos conectados por WIFI, para estudiar los efectos causados por uno o varios usuarios compitiendo por el mismo canal, por ejemplo, unos tratando de subir datos mientras otros tratan de bajar datos.
- Debido a la proliferación de dispositivos que imitan a la Raspberry Pi[1], sería interesante analizar las capacidades de otros dispositivos que no sean ODROID.
- Estudiar si haciendo alguna mejora adicional el algoritmo de detección de retransmisiones por huecos baja sus tasas de error.
- Realizar más pruebas con los algoritmos de detección de retransmisiones, con tráfico más variado y no solo con tráfico HTTP.

[1] `https://www.raspberrypi.org/`

Bibliografía

[1] V. Moreno, P. M. Santiago del Río, J. Ramos, D. Muelas, J. L. García-Dorado, F. J. Gómez-Arribas, and J. Aracil, "Multi-granular, multi-purpose and multi-Gb/s monitoring on off-the-shelf systems," *International Journal of Network Management*, vol. 24, no. 4, pp. 221–234, 2014.

[2] J. Ramos, P. S. del Río, J. Aracil, and J. E. López de Vergara, "On the effect of concurrent applications in bandwidth measurement speedometers," *Computer Networks*, vol. 55, no. 6, pp. 1435 – 1453, 2011.

[3] J. Salvachua, J. Garcia, J. Quemada, M. Cortes, L. Vizcaino, G. Fernandez, J. Lajo, and C. Barcenilla, *Internet Draft: The Quality for Service Protocol*, Std., 2015.

[4] C. Demichelis and P. Chimento, *RFC 3393: IP packet delay variation metric for IP performance metrics (IPPM)*, Std., 2002.

[5] J. Fabini and A. Morton, *Advanced Stream and Sampling Framework for IPPM*, Std., 2014.

[6] G. Almes, M. Zekauskas, S. Kalidindi, and A. Morton, *RFC 7679: A One-Way Delay Metric for IP Performance Metrics (IPPM)*, Std., 2016.

[7] G. Almes, S. Kalidindi, and M. Zekauskas, *RFC 7680: A one-way packet loss metric for IP Performance Metrics (IPPM)*, Std., 2016.

[8] D. V. Schuehler and J. W. Lockwood, "A modular system for fpga-based tcp flow processing in high-speed networks," in *International Conference on Field Programmable Logic and Applications*. Springer, 2004, pp. 301–310.

[9] R. Leira, P. Gomez, I. Gonzalez, and J. L. de Vergara, "Multimedia flow classification at 10 Gbps using acceleration techniques on commodity hardware," in *Smart Communications in Network Technologies (SaCoNeT), 2013 International Conference on*, vol. 3. IEEE, 2013, pp. 1–5.

[10] M. Forconesi, G. Sutter, S. Lopez-Buedo, J. E. López de Vergara, and J. Aracil, "Bridging the gap between hardware and software open source network developments," *IEEE Network*, vol. 28, no. 5, pp. 13–19, September 2014.

[11] P. Roquero, J. Ramos, V. Moreno, I. González, and J. Aracil, "High-speed TCP flow record extraction using GPUs," *The Journal of Supercomputing*, vol. 71, no. 10, pp. 3851–3876, 2015.

[12] V. Moreno, J. Ramos, P. M. Santiago del Río, J. L. García-Dorado, F. J. Gómez-Arribas, and J. Aracil, "Commodity Packet Capture Engines: Tutorial, Cookbook and Applicability," *IEEE Communications Surveys Tutorials*, vol. 17, no. 3, pp. 1364–1390, 2015.

[13] V. Moreno, J. Ramos, J. L. García-Dorado, I. Gonzalez, F. J. Gomez-Arribas, and J. Aracil, "Testing the capacity of off-the-self systems to store 10GbE traffic," *IEEE Communications Magazine*, 2015.

[14] B. Claise, B. Trammell, and P. Aitken, *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information RFC 7011*, Std., 2013.

[15] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A Survey on Software-Defined Networking," *IEEE Communications Surveys Tutorials*, vol. 17, no. 1, pp. 27–51, 2015.

[16] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[17] E. Hernandez-Valencia, S. Izzo, and B. Polonsky, "How will nfv/sdn transform service provider opex?" *IEEE Network*, vol. 29, no. 3, pp. 60–67, 2015.

[18] E. Miravalls Sierra, "Automatización y detección de anomalías en tráfico de Internet," Trabajo Fin de Grado, 2016. [Online]. Available: https://repositorio.uam.es/bitstream/handle/10486/676220/Miravalls_Sierra_Eduardo_tfg.pdf

[19] I. Ucar, D. Morato, E. Magaña, and M. Izal, "Duplicate detection methodology for ip network traffic analysis," in *2013 IEEE International Workshop on Measurements Networking (M N)*, Oct 2013, pp. 161–166.

[20] R. Bejtlich, *The practice of network security monitoring: understanding incident detection and response*. No Starch Press, 2013.

[21] C. Sanders and J. Smith, *Applied network security monitoring: collection, detection, and analysis*. Elsevier, 2013.

[22] V. Moreno, P. M. Santiago del Rio, J. Ramos, J. L. Garcia-Dorado, I. Gonzalez, F. J. Gomez-Arribas, and J. Aracil, "Packet storage at multi-gigabit rates using off-the-shelf systems," in *16th IEEE International Conference on High Performance Computing and Communications (HPCC2014)*, August 2014.

[23] L. Rizzo, L. Deri, and A. Cardigliano, "10 gbit/s line rate packet processing using commodity hardware: Survey and new proposals," 2012.

[24] L. Rizzo, "Netmap: a novel framework for fast packet i/o," in *21st USENIX Security Symposium (USENIX Security 12)*, 2012.

[25] N. Bonelli, S. Giordano, and G. Procissi, "Network traffic processing with PFQ," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 6, pp. 1819–1833, 2016.

[26] V. Moreno, "Harnessing low-level tuning in modern architectures for high-performance network monitoring in physical and virtual platforms," Ph.D. dissertation, Universidad Autónoma de Madrid, May 2015. [Online]. Available: http://arantxa.ii.uam.es/~vmoreno/Publications/morenoPhD2015.pdf

[27] G. Julián Moreno, "Monitorización, captura y almacenamiento inteligente de tráfico de red a 40Gbps," Trabajo Fin de Grado, 2016. [Online]. Available: https://repositorio.uam.es/bitstream/handle/10486/675469/Julian_Moreno_Guillermo_tfg.pdf

[28] D. Intel, "Intel Data Plane Development Kit," *Programmer's Guide*, 2016. [Online]. Available: http://dpdk.org/doc/guides/prog_guide

[29] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis, *RFC 2330: Framework for IP performance metrics*, Std., 1998.

[30] H. Hedayat, R. Krzanowski, A. Morton, K. Yum, and J. Babiarz, *RFC 5357: A two-way active measurement protocol*, Std., 2008.

[31] E. Atxutegi, F. Liberal, E. Saiz, and E. Ibarrola, "Toward standardized internet speed measurements for end users: current technical constraints," *IEEE Communications Magazine*, 2016.

[32] C. Pei, Y. Zhao, G. Chen, R. Tang, Y. Meng, M. Ma, K. Ling, and D. Pei, "Wifi can be the weakest link of round trip network latency in the wild," in *35th Annual IEEE International Conference on Computer Communications (INFOCOM)*, 2016.

[33] Z. Hu, Y.-C. Chen, L. Qiu, G. Xue, H. Zhu, N. Zhang, C. He, L. Pan, and C. He, "An in-depth analysis of 3G traffic and performance," in *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*. ACM, 2015.

[34] M. Maity, B. Raman, and M. Vutukuru, "Tcp download performance in dense wifi scenarios," in *2015 7th International Conference on Communication Systems and Networks (COMSNETS)*, 2015.

[35] J. Strauss and M. F. Kaashoek, "Estimating bulk transfer capacity."

[36] J. R. de Santiago and J. A. Rico, "Proactive measurement techniques for network monitoring in heterogeneous environments," Ph.D. dissertation, Ph. D. dissertation, Universidad Autónoma de Madrid, 2013. [Online]. Available: https://repositorio.uam.es/bitstream/handle/10486/14336/67011_ramos_de_santiago_javier.pdf

[37] G. Almes, M. J. Zekauskas, and S. Kalidindi, *RFC 2681: A round-trip delay metric for IPPM*, Std., 1999.

[38] J. Mahdavi and V. Paxson, *RFC 2678: IPPM metrics for measuring connectivity*, Std., 1999.

[39] E. Charfi, L. Chaari, and L. Kamoun, "PHY/MAc enhancements and QoS mechanisms for very high throughput WLANs: A survey," *IEEE Communications Surveys & Tutorials*, 2013.

[40] R. Karmakar, S. Chakraborty, and S. Chattopadhyay, "Impact of IEEE 802.11 n/ac PHY/MAC high throughput enhancements over transport/application layer protocols-a survey," *arXiv preprint:1702.03257*, 2017.

[41] S. Basso, M. Meo, A. Servetti, and J. C. De Martin, "Estimating packet loss rate in the access through application-level measurements," in *Proceedings of the 2012 ACM SIGCOMM Workshop on Measurements Up the Stack*, ser. W-MUST '12. New York, NY, USA: ACM, 2012, pp. 7–12.

[42] Y.-C. Chen, Y.-s. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley, "A measurement-based study of multipath tcp performance over wireless networks," in *Proceedings of the 2013 Conference on Internet Measurement Conference*, ser. IMC '13, 2013, pp. 455–468.

[43] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub, "Extrapolation Methods for Accelerating PageRank Computations," in *Proceedings of the 12th International Conference on World Wide Web*, ser. WWW '03. ACM, 2003, pp. 261–270. [Online]. Available: http://doi.acm.org/10.1145/775152.775190

[44] J. L. Garcia-Dorado, J. A. Hernandez, J. Aracil, J. E. L. de Vergara, F. J. Monserrat, E. Robles, and T. P. de Miguel, "On the duration and spatial characteristics of internet traffic measurement experiments," *IEEE Communications Magazine*, vol. 46, no. 11, pp. 148–155, November 2008.

[45] N. Brownlee and K. C. Claffy, "Understanding Internet traffic streams: dragonflies and tortoises," *IEEE Communications Magazine*, vol. 40, no. 10, pp. 110–117, 2002.

[46] M. Newman, "Power laws, Pareto distributions and Zipf's law," *Contemporary Physics*, vol. 46, no. 5, pp. 323–351, 2005.

[47] J. D. Little and S. C. Graves, "Little's law," in *Building intuition*. Springer, 2008, pp. 81–100.

[48] R. Hofstede, I. Drago, A. Sperotto, R. Sadre, and A. Pras, *Measurement Artifacts in NetFlow Data*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–10.

[49] W. R. Stevens, *RFC 1122: TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms*, Std., 1997.

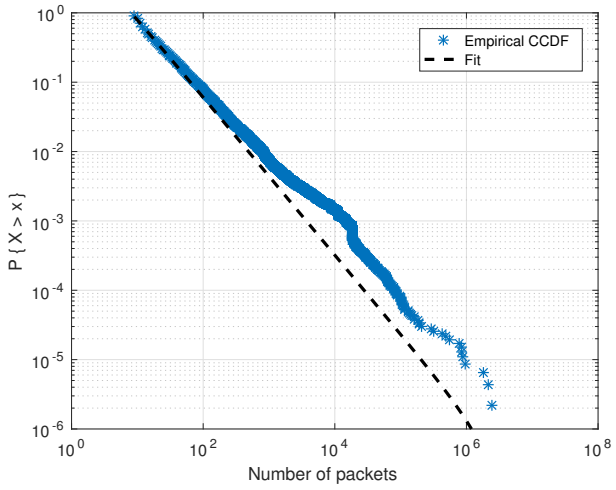
[50] D. M. Powers, "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation," 2011.

Apéndices

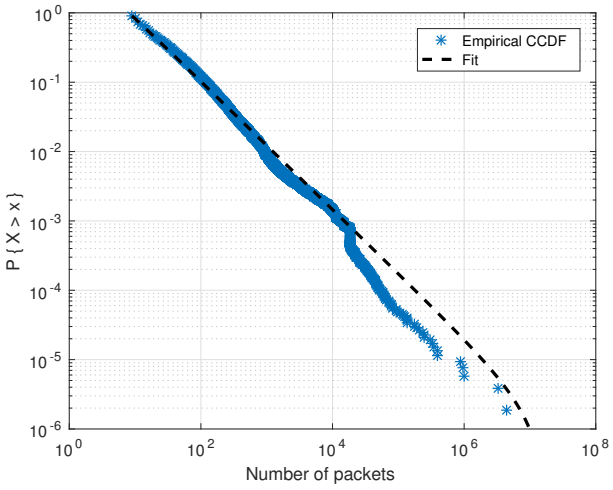
A

Distribuciones del número de paquetes por flujo en las trazas A,B,C

A continuación se muestran las funciones de supervivencia (CCDF) de los tamaños de flujo de los conjuntos de datos utilizados en el Capítulo 5.

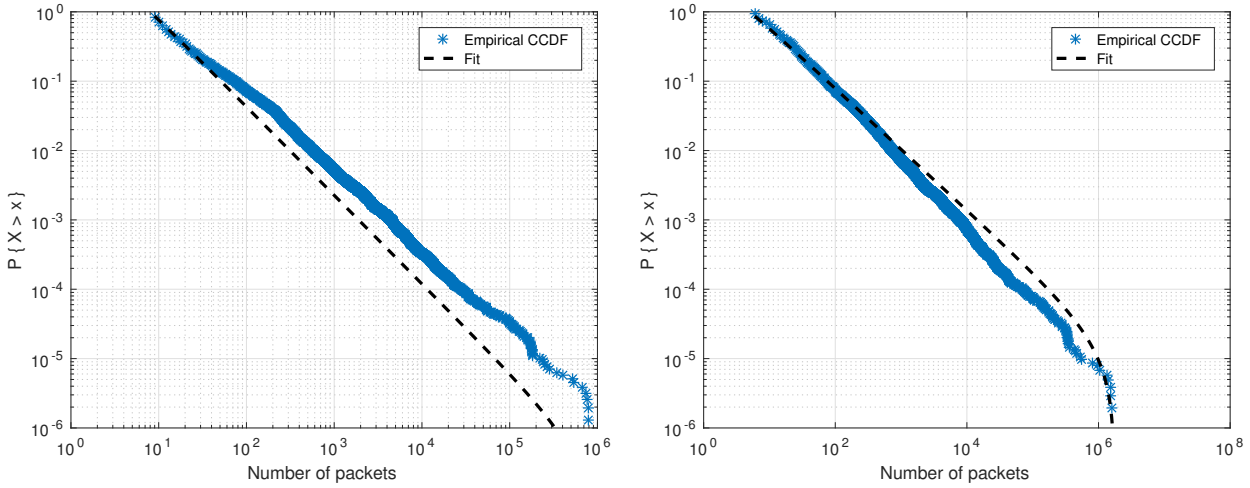


(a) Cliente a Servidor



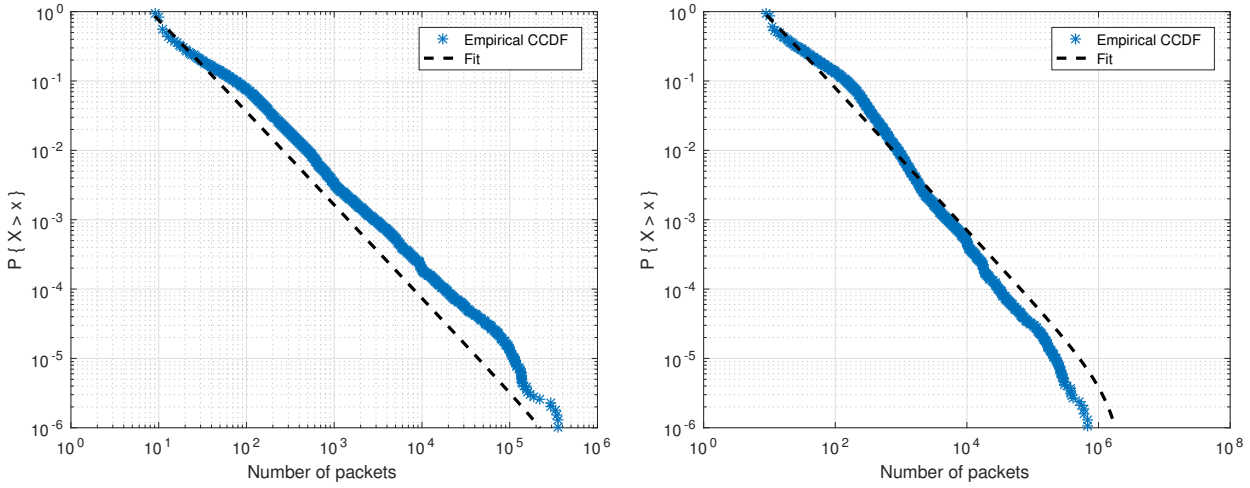
(b) Servidor a Cliente

Figura A.1: Distribución del número de paquetes por flujo de la traza A.



(a) Cliente a Servidor (b) Servidor a Cliente

Figura A.2: Distribución del número de paquetes por flujo de la traza B.



(a) Cliente a Servidor (b) Servidor a Cliente

Figura A.3: Distribución del número de paquetes por flujo de la traza C.

B

Resultados derivados

Los resultados derivados de este Trabajo Fin de Máster se han incluido en las siguientes publicaciones:

- Sección 2.2 y Capítulo 3: "Evaluación de equipamiento de bajo coste para realizar medidas de red en entornos domésticos". Eduardo Miravalls-Sierra, David Muelas, Jorge E. López de Vergara, Javier Ramos, Javier Aracil. Enviado a las Jornadas de la Ingeniería Telemática (JITEL) 2017.
- Capítulo 5: "Online Detection of Pathological TCP Flows with Retransmissions in High-speed Networks". Eduardo Miravalls-Sierra, David Muelas, Javier Ramos, Jorge E. López de Vergara, Daniel Morató, Javier Aracil. Enviado a Computer Communications ISSN: 0140-3664, y pendiente de revisión.
- Capítulo 2, Capítulo 3 y Capítulo 4 forman parte de los entregables 5.1.1 y 5.2.1 del proyecto nacional Racing Drones (MINECO / FEDER RTC-2016-4744-7).

Además, las aplicaciones industriales han sido:

- La sección de evaluación del Capítulo 3 se ha usado para validar el cliente que se utiliza en el proyecto de Escuelas Conectadas.
- El *algoritmo basado en desorden* se ha implementado en una aplicación de análisis de tráfico comercial, Detect-Pro, desarrollada por Naudit que va a ser desplegada en datacenters de varias grandes multinacionales.